Uma Plataforma de Hardware para Processamento de Imagem baseada na Transformada Imagem-Floresta

Fábio Augusto Menocci Cappabianco

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Uma Plataforma de Hardware para Processamento de Imagem baseada na Transformada Imagem-Floresta

Fábio Augusto Menocci Cappabianco [2]

15 de fevereiro de 2006

Banca Examinadora:

- Prof. Dr. Guido Costa Souza de Araújo
  Instituto de Computação - Universidade Estadual de Campinas (Orientador)
- Prof. Dr. Rodolfo Jardim de Azevedo
  Instituto de Computação - Universidade Estadual de Campinas
- Prof. Dr. Roberto Marcondes Cesar Junior
  Departamento de Ciência de Computação - IME - Universidade de São Paulo
- Prof. Dr. Paulo Cesar Centoducatte
  Instituto de Computação - Universidade Estadual de Campinas

[2]**Financiado pela Fundação de Amparo a Pesquisa do estado de São Paulo (FAPESP), processo número 03/11673-4**

Uma Plataforma de Hardware para Processamento de Imagem baseada na Transformada Imagem-Floresta

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fábio Augusto Menocci Cappabianco [3] e aprovada pela Banca Examinadora.

Campinas, 15 de fevereiro de 2006.

Prof. Dr. Guido Costa Souza de Araújo
Instituto de Computação - Universidade Estadual de Campinas (Orientador)

Prof. Dr. Alexandre Xavier Falcão
(Co-orientador)
Instituto de Computação - Universidade Estadual de Campinas

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Substitua pela folha com a assinatura da banca

[4]Financiado pela Fundação de Amparo a Pesquisa do estado de São Paulo (FAPESP), processo número 03/11673-4

Agradecimentos

Primeiramente a Jeová Deus pelo dom da vida e pelo trabalho satisfatório.

Aos orientadores Prof. Dr. Guido Araújo e Prof. Dr. Alexandre Falcão pelas diretrizes principais, sem as quais o projeto não se concretizaria.

À minha família, Oduvaldo, Márcia, César, Renan, Therezinha, Olimar, Umbelina, Germinal, minhas primas, tios e bisavôs, Carolline, Juliana, Guilherme, Geraldo, Irene, pela proximidade, amor, carinho e apoio.

Aos amigos, cosiderados como irmãos, Thiago, Kleber e Bruno, Thais e Mozart, Fabiano e Cristina, Mateus, Isaac, Carlos e Alessandra, Junior e Marina, Queimson, André e Camila, Daniel, Marco, Rodolfo, Ricardo, e aos demais amigos de Campinas e Franca.

A todos os colegas da academia, professores e funcionários do Instituto de Computação da UNICAMP que apoiaram de algum modo o projeto, em especial ao professor Elmar Melcher da Universidade Federeal de Campina Grande.

A FAPESP pelos recursos que permitiram o desenvolvimento da pesquisa.

Resumo

Implementações de operadores de processamento de imagens em plataformas de hardware têm obtido ótimos resultados devido a sua atuação paralela em diversas regiões da imagem. Ao mesmo tempo, a IFT (*Image Foresting Transform*) tem provado ser uma técnica eficiente de reduzir problemas de processamento de imagens em um problema de floresta de caminhos de um grafo, cuja solução é obtida em tempo linear no o número de pixels. Este trabalho contém a implementação de uma plataforma, em hardware, chamada SIFT (*Silicon Image Foresting Transform*), que executa o algoritmo da IFT paralelamente. O modelo de processamento e armazenamento SIFT serve como base para outras arquiteturas de processamento de imagens e amplia o entendimento de alguns conceitos de mapas de predecessores e rótulos utilizados pela IFT.

Abstract

Great results had been achieved by the use of hardware platforms to implement image processing operators. This success was reached due to the use of multiple processors working parallel in several regions of the image. On the other hand, IFT (*Image Foresting Transform*), a software technique to reduce image processing problems into a graph path forest problem, performs image operations in linear time in the number of pixels in most of applications. The main goal of this work was to generate a hardware platform, that implements the an algorithm based on the IFT in a fast and efficient way.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Uma imagem pode ser representada de forma natural ou implícita como um grafo, no qual os pixels são os nós e os arcos são definidos por uma relação de adjacência entre pixels. Neste contexto, um operador de imagem pode ser interpretado como um percurso no grafo seguido de uma operação local simples sobre o grafo resultante deste percurso. Esta abordagem tem como principal vantagem a variedade de algoritmos eficientes em grafos com aplicação direta em processamento de imagens. Um exemplo do seu potencial é a Transformada Imagem-Floresta (*Image Foresting Transform*-IFT). A IFT é uma técnica recente de reduzir problemas de processamento de imagens em um problema de floresta de caminhos de um grafo cuja solução é obtida em tempo linear com o número de pixels na maioria das aplicações [1–6].

A IFT unifica a solução de vários problemas em um algoritmo genérico; permite compreender de forma natural a relação entre alguns operadores de imagem (e.g. transformada de *watershed* e reconstrução morfológica [7]); estabelece critérios no projeto de operadores que garantem a corretude da solução de um problema [1]; provê mecanismos para modificar alguns operadores sem que estes percam suas propriedades originais [2]; oferece ganho de eficiência consideráveis na maioria das operações (e.g. transformada de distância Euclideana e operadores relacionados [8–10]); e possui diversas aplicações em filtragem, segmentação e análise de imagens (e.g. transformadas de *watershed* [2, 11], reconstrução morfológica e operadores relacionados [7], esqueletos multi-escala e transformada de distância Euclideana [8,9,12], dimensão fractal multi-escala e pontos de saliência de curvas [10], perseguição de bordas [13–15]).

Viabilizar implementações velozes de algoritmos de processamento de imagem é, ainda, um problema complexo. Apesar disso, o grande crescimento das indústrias de silício (*foundries*) e o refinamento de suas tecnologias tem possibilitado a integração de milhões de portas lógicas dentro de única pastilha de silício. Circuitos de grande complexidade irão permitir o projeto de sistemas inteiros em um único chip, uma metodologia de projeto

VLSI que está sendo chamada *System-on-a-Chip* (SoCs).

Ao passo que a tecnologia de SoCs abre inúmeras possibilidades para o projeto de sistemas digitais, por outro lado, tem dificultado a vida dos projetistas destes sistemas. O tempo e os recursos necessários durante o desenvolvimento destes vem aumentando exponencialmente devido à complexidade destas arquiteturas. Portanto, é de se esperar que metodologias de projeto baseadas em ASIC (*Appllication Specific Integrated Circuit*) sofram alterações no futuro próximo, caso contrário se tornarão inviáveis.

Circuitos pré-validados que possam ser moldados a uma aplicação específica, conhecidos como FPGAs (*Field Programmable Gate Arrays*), estão sendo vislumbrados como a alternativa tecnológica para se contornar os problemas de projeto com esta complexidade. Estes circuitos, podem ser programados de acordo com a aplicação do usuário através da compilação de um programa que descreve a arquitetura, o que é bem mais simples comparando-se à metodologia ASIC.

Independentemente da metodologia utilizada em hardware, plataformas de hardware que implementam operadores de processamento de imagens têm obtido ótimos resultados [16–18]. O sucesso se dá, principalmente, devido à localidade (vizinhança) e à similaridade das operações realizadas pelos mesmos. Desta forma, múltiplos processadores podem atuar paralelamente em diversas regiões da imagem em uma plataforma rápida e eficiente.

Nestas circunstâncias, uma plataforma em hardware chamada SIFT foi idealizada e implementada, baseando-se no algoritmo da IFT, na qual diversos operadores de imagem podem ser executados. A SIFT reune as vantagens da baixa complexidade e do uso genérico da IFT com a velocidade resultante da sua implementação paralela em hardware.

Dois modelos diferentes da SIFT foram gerados: o modelo global, apresentando-se com solução mais simples e rápida, demandando, porém mais recursos de hardware; e o modelo setorial, que também se mostrou muito mais rápido que a IFT original e pode ser adaptado aos recursos de hardware disponíveis.

1.1 Objetivos

Os objetivos deste projeto foram:

- Pesquisa e desenvolvimento de algoritmos paralelos que aumentassem o desempenho de técnicas de processamento de imagens baseadas na Transformada Floresta-Imagem.
- Implementação dos algoritmos em FPGA's.
- Avaliação da plataforma projetada em aplicações de alto-desempenho como segmentação e visualização.

Como principais contribuições durante o período de desenvolvimento obteve-se:

- Uma plataforma de processamento de imagens em uma FPGA.
- Uma nova implementação para o algoritmo da IFT com uma breve explanação de sua corretude.
- Novo ponto de vista e maiores esclarecimentos de conceitos utilizados pela IFT.
- Uma maneira elegante para armazenar dados de imagens processadas em hardware utilizando paralelismo de imagem e setorial.

O restante desta dissertação está dividida do seguinte modo: O Capítulo 2 contém um resumo a respeito de algumas arquiteturas de hardware utilizadas para processamento de imagens; o Capítulo 3 contém uma breve explicação da representação de uma imagem digital como um grafo e suas vantagens; o Capítulo 4 contém uma base teórica e algumas aplicações da ferramenta da transformada imagem-floresta (IFT); o Capítulo 5 apresenta o modelo, implementação e aplicações da arquitetura de hardware da IFT (SIFT); e o Capítulo 6 demonstra os resultados, conclusões e trabalhos futuros do mestrado.

Capítulo 2

Técnicas de Processamento de Imagens em Hardware

Muito progresso tem sido alcançado pelos computadores e hardwares que possibilitam o processamento e análise cada vez mais veloz, complexa e eficiente das imagens.

Este capítulo contém uma breve revisão histórica do desenvolvimento de hardware para processamento de imagens, uma descrição da maneira como o hardware deve ser desenvolvido para processar imagens, um exemplo de plataforma desenvolvida e o desafio encontrado para entrada, saída e armazenamento de grande quantidade de dados.

2.1 Histórico do desenvolvimento do hardware

Nos anos 60, surgiu o primeiro computador poderoso suficiente para armazenar e processar uma imagem significativa. Desde então, avanços maiores foram atingidos com a criação das linguagens de programação de alto nível como FORTRAN E COBOL (1960), o desenvolvimento do microprocessador, como o modelo da CPU, memória e periféricos (1970), o computador pessoal em 1981 pela IBM e a progressiva miniaturização de componentes, começando com a integração de grande escala (LI) nos anos 60, integração de muito grande escala (VLSI) nos anos 70 e atualmente ultra grande escala (ULSI).

Ao longo do tempo, este grande crescimento da densidade de transistores em sistemas têm possibilitado a integração de milhões de portas lógicas dentro de uma única pastilha de silício. Transistores MOS (*Metal-Oxide-Semiconductor*) com comprimento de canal da ordem de 130nm têm permitido o projeto de circuitos VLSI (*Very Large Scale Integration*) contendo mais de 10 Milhões de portas lógicas. A partir do ano de 2006, com o advento da tecnologia de 65nm, abre-se a possibilidade de que circuitos contendo até 370 Milhões de

transistores possam ser implementados[1]. Circuitos com esta complexidade irão permitir o projeto de sistemas inteiros em um único chip, uma metolodologia de projeto VLSI que está sendo chamada *System-on-a-Chip* (SoCs).

A tecnologia de SoCs abre inúmeras possibilidades para o projeto de sistemas digitais, mas, ao mesmo tempo, está criando um pesadelo para os projetistas destes sistemas. A complexidade destas arquiteturas vem aumentando exponencialmente o tempo necessário para o projeto destes sistemas. Em um exemplo típico, o chip PNX-8500 de Philips [19], um ASIC (*Appllication Specific Integrated Circuit*) utilizado na descompressão de imagens e operações de *set-top box*, demandou 60 engenheiros durante 2 anos para ficar pronto [19], sendo que 70% dos recursos foram gastos somente na tarefa de verificação funcional do chip. Dado que este chip possui 8 Milhões de portas lógicas em uma tecnologia de 180nm, um SoC equivalente que faça uso de apenas 30% das 100 milhões de portas lógicas que estarão disponíveis na tecnologia de 90nm, utilizando-se os mesmos métodos, demandaria, proporcionalmente 180 engenheiros nos mesmos dois anos, um projeto em uma escala nunca vista antes. A conclusão óbvia é que para sistemas desta complexidade, metodologias de projeto baseadas em ASIC são inviáveis.

A alternativa tecnológica que está sendo vislumbrada para se contornar os problemas de projeto com esta complexidade é a utilização de circuitos pré-validados que possam ser moldados a uma aplicação específica. Estes circuitos, conhecidos como FPGAs (*Field Programmable Gate Arrays*), possuem milhões de portas lógicas pré-fabricadas e validadas, cujas interconexões podem ser programadas via software de acordo com a aplicação do usuário, de modo que a arquitetura resultante execute somente a funcionalidade desejada.

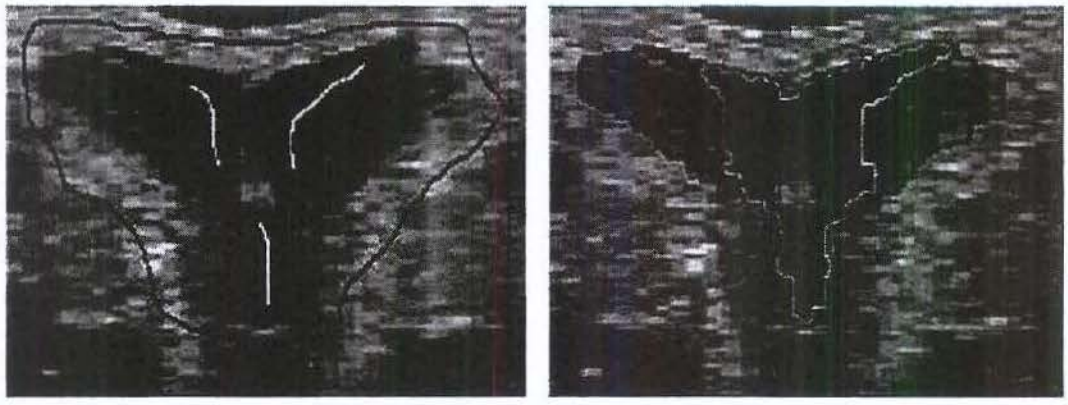
A programação de uma FPGA é feita através da compilação de um programa que descreve a arquitetura. Este programa é escrito usando HDL (*Hardware Description Language*)[2] como VHDL [20] ou Verilog [21], que surgiram em meados dos anos 80. Compiladores para VHDL e Verilog capazes de gerar circuitos para FPGAs são disponibilizados por companhias fabricantes de FPGA como Xilinx e Altera.

2.2 Desenvolvimento de arquiteturas de processamento de imagens em hardware

Inúmeras implementações de algoritmos em hardware para processamento e análise de imagens já foram desenvolvidas [16–18]. O sucesso destas implementações se dá, principalmente, devido à localidade (vizinhança) e à similaridade das operações realizadas.

Uma grande quantidade de operadores realizam processamento para os pixels baseando-

[1]Para mais detalhes veja o site da Intel em http://www.intel.com/technology/silicon/65nm_technology.htm
[2]Do Inglês: Linguagem de Descrição de Hadware



(a) (b)

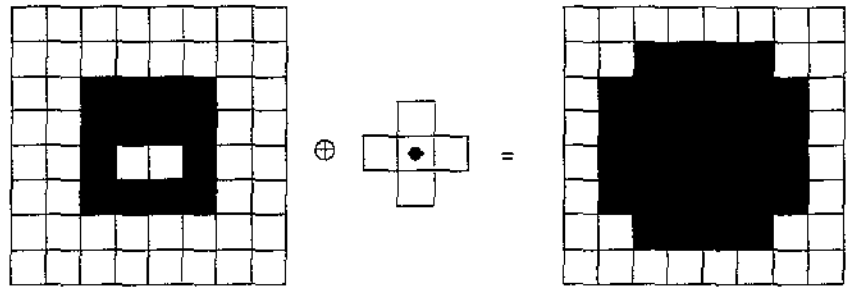
Figura 2.1: Exemplo de transformada de Watershed sobre imagem de núcleo caudado do cérebro humano. (a) Imagem original do cérebro com furos de bacia nas linhas coloridas. (b) Imagem segmentada depois de aplicada a transformada de Watershed.

se apenas em dados locais. A transformada de Watershed, por exemplo, utilizada para segmentação e rotulação de imagens [2,22–25], é um operador no qual os pixels da imagem que representam pontos de uma região de relevo são "inundados" por pontos adjacentes. A Figura 2.1 contém um exemplo da transformada de Watershed sobre uma imagem de ressonância magnética do cérebro humano. As reconstruções morfológicas [7,26–28] também são operadores que utilizam máscaras que restringem a influência entre pixels da imagem a uma proximidade. A Figura 2.2 contém exemplos de uma dilatação e uma erosão, que são operações morfológicas básicas.

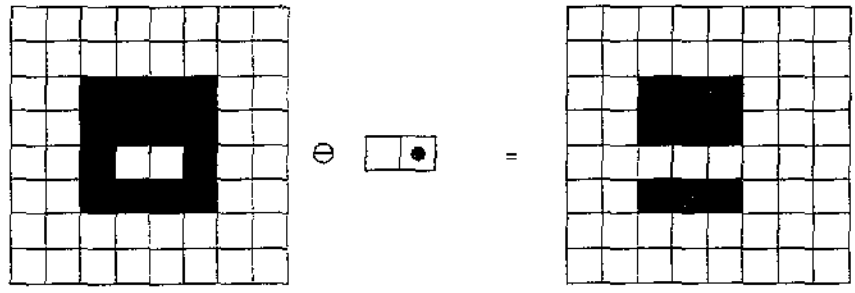
Aproveitando-se das vantagens da localidade e igualdade dos operadores para todos os pixels, múltiplos elementos processadores podem atuar paralelamente em diversas regiões da imagem em uma plataforma rápida e eficiente. De acordo com a pesquisa de Danielson [29], existem tipos de paralelismo de arquiteturas para o processamento de imagens. O tipo interessante para um processamento genérico como a IFT [1] é o paralelismo de imagem que consiste em dividir os pixels da imagem de tal modo que exista um processador para cada uma das regiões segmentadas.

Um computador paralelo consiste em um sistema computacional contendo múltiplos processadores que se comunicam e cooperam para resolver grandes problemas mais rapidamente [30]. Este computador paralelo pode operar em quatro modos distintos, classificados de acordo com os fluxos de dados e instruções como definido em [31]:

1. SISD (Single Instruction Stream, Simgle Data Stream);



(a)



(b)

Figura 2.2: Exemplo de transformações morfológicas sobre imagens. (a) Imagem original dilatada por máscara em forma de cruz. (b) Imagem original erodida por máscara em forma de linha.

2. SIMD (Single Instruction Stream, Multiple Data Stream);

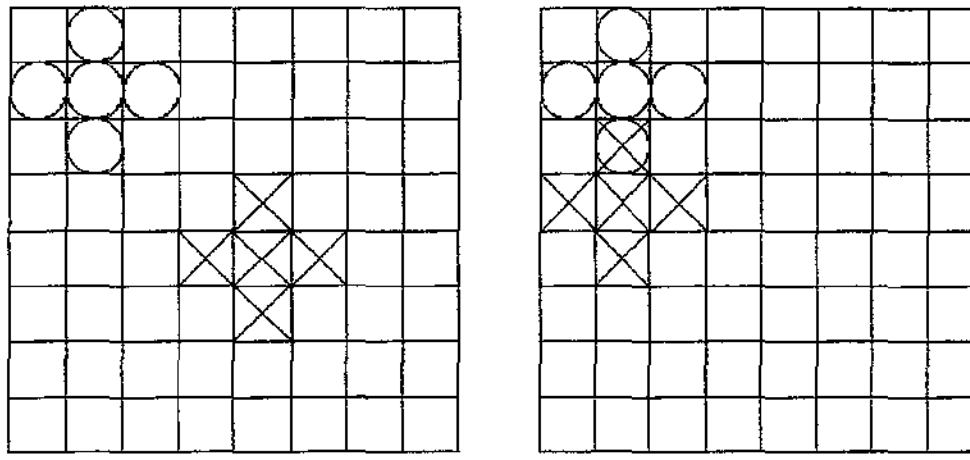
3. MISD (Multiple Instruction Stream, Single Data Stream);

4. MIMD (Multiple Instruction Stream, Multiple Data Stream);

Nesta classificação *SINGLE INSTRUCTION STREAM* significa que os processadores são alimentados por um único fluxo de instruções e *MULTIPLE INSTRUCTION STREAM* significa que são alimentados por múltiplos fluxos. O mesmo ocorre no caso das classificações *SINGLE DATA STREAM* e *MULTIPLE DATA STREAM*, sendo, neste caso, os fluxos referentes aos dados que alimentam os processadores.

No caso dos operadores de imagem citados anteriormente, como Watershed e transformações morfológicas, pode-se utilizar um computador paralelo operando no modo

MIMD, desde que mais de um processador não atue sobre o mesmo pixel ou em vizinhanças com interseções, para que não haja inconsistências em leituras ou escritas. A Figura 2.3 ilustra um uso correto e um incorreto de processamento paralelo.



(a) (b)

Figura 2.3: Processamento paralelo de imagem por dois processadores. Um processador utiliza dados dos pixels marcados por 'x' e outro dos pixels marcados por 'o'. Na Figura (a), o processamento produz um resultado correto. Não há interferência entre os processadores. Em (b), porém, pode haver inconsistência entre os processadores, pois ambos utilizam dados de um mesmo pixel ao mesmo tempo.

2.3 Exemplo de arquitetura desenvolvida

Um trabalho de processamento de imagens utilizando processamento paralelo em modo MIMD foi realizado em [18]. Nele, a imagem é rotulada através do método da transformada de Watershed. O propósito deste operador é de segmentar objetos pelos rótulos distintos.

A idéia desta implementação é de, primeiramente, dividir a imagem em regiões menores (seções) e calcular a transformada de Watershed em cada uma com um processador diferente. Como resultado da transformada de Watershed, obtêm-se diversos platôs de pixels. O algoritmo rotula todos os pixels de um mesmo platô com o mesmo rótulo.

Para cada platô de todas as seções, associa-se, então, um nó de um grafo contendo

o rótulo do mesmo. Posteriormente, faz-se uma avaliação dos nós das bordas de setores vizinhos, unificando os rótulos dos nós do grafo referentes aos platôs que contém os nós coexistentes nas bordas dos setores. Obtem-se como resultado o mapa de rótulos da imagem original.

O método gera um resultado correto porque o mapa de rótulos não leva em conta o caminho pelo qual a transformada percorre para atingir os pixels. Sendo assim, sabendo apenas que uma região foi inundada a partir da mesma origem, não importando de onde esta origem se encontra, é possível associar partes deste caminho ao mesmo label, antes mesmo deste ser alcançado por ela. Assim, cada processador associado a um setor pode operar com fluxo de dados locais, sem interferência de outros processadores e com fluxo de instruções próprias, pois a ordem das operações não altera o resultado.

2.4 Desafios na transmissão e armazenamento dos dados

A maneira com que os dados entram e saem de uma plataforma depende da interface externa de comunicação. Nos computadores pessoais, por exemplo, são utilizadas mídias removíveis, como CDROM's e DVDROM's, portas de comunicação serial e paralela, discos locais e muitos outros. Outras plataformas como as FPGA's não possuem discos locais e, dependendo do modelo, podem ser acessadas por portas seriais, paralelas, cartões de memória e barramentos PCI.

Encontrar uma interface que atenda aos requisitos do sistema é um grande desafio, principalmente quando uma grande quantidade de dados é requerida em pouco tempo. Este é o caso do processamento de imagens.

Quando o processamento é realizado em software, o problema de entrada e saída é inteiramente do sistema operacional. O desenvolvedor preocupa-se apenas com o armazenamento, utilizando uma estrutura de dados eficiente para que o acesso seja rápido, exija poucas operações matemáticas (como o cálculo de índices de vetores), uso de apontadores eficiente e outras otimizações. O algoritmo da IFT em software é um ótimo exemplo no qual os dados são acessados por um único índice de memória, com tabelas de posições de colunas pré calculadas e uma fila de prioridade extremamente rápida[3].

Já o processamento em uma FPGA tem um desafio diferente. Além da escolha da interface a ser utilizada, normalmente exige-se que um protocolo para a comunicação seja desenvolvido. Alguns kits de desenvolvimento como o NIOS-II da Altera[4] já possuem

[3]O código do algoritmo da IFT está disponível no endereço http://www.ic.unicamp.br/ãfalcao/download/gift.tar.gz

[4]Para maiores detalhes veja o site da Altera em http://www.altera.com/products/devkits/kit-dev_platforms.jsp

uma interface pronta e eficiente, para diversos dispositivos. No entanto, como a entrada e saída de dados é uma parte crítica de plataformas de processamento de imagens devido às grandes matrizes de pixels, interfaces específicas que são mais rápidas se tornam mais atraentes. Além disso, como o processamento de imagens em hardware envolve um grande número de instâncias de processadores para o paralelismo, o espaço utilizado também é crítico. Plataformas prontas como o NIOS-II, mesmo reduzidas, ocupam um espaço razoável dentro das FPGA's, enquanto que interfaces específicas geralmente são bem menores.

Quanto ao armazenamento de dados FPGA's, existe uma grande diversidade de tipos de memória interna e externa, utilizáveis. O kit de desenvolvimento Nios-II com FPGA Stratix-II modelo EP2S60F672C5, por exemplo, utiliza três tipos de memória RAM internas ao chip, de tamanhos e barramentos de largura diferentes (M512RAM, M4KRAM e M-RAM), além de elementos lógicos. O kit possui também memórias SRAM, SDRAM e flash externas à FPGA mas dentro da placa, sendo de rápido acesso [32]

A maneira como os dados são armazenados na FPGA está diretamente relacionada com o tempo de processamento da mesma. Se os dados demoram para ser buscados na memória, o processamento fica paralisado. Portanto, é importante que a memória seja bem estruturada.

Uma das contribuições deste trabalho é de estender para hardware operadores locais de processamento de imagens baseados na IFT. Mas, para isso, a imagem precisa ser representada como um grafo, como descrito no próximo capítulo.

Capítulo 3

Imagem em Grafo

Montanari [33] e Martelli [34,35] foram uns dos primeiros pesquisadores a formular problemas associados ao processamento de imagens em problemas de caminho mínimo em grafos. A partir de então, diversos autores tiveram contribuições neste sentido, tornando-se apropriada a interpretação da imagem como um grafo [1,13,22,36,37].

Neste capítulo serão abordadas as maneiras que uma imagem pode ser interpretada como um grafo. A Seção 3.1 traz o conceito de vértices e arestas em uma imagem, a Seção 3.2 contém uma breve explanação a respeito de caminhos, conexidade e particionamento de imagens, a Seção 3.3 explica o que são custos de caminho, funções de custo de caminho, distância entre pixels e relaciona estes conceitos com o particionamento de imagens e a Seção 3.4 conceitua mapas de predecessores, associando-os com o conceito de florestas e florestas de caminhos mínimos.

3.1 Vértices e arestas em imagens

Diversas estratégias foram desenvolvidas para processar a imagem através de sua representação como um grafo. Em algumas delas, os vértices do grafo representam regiões ou objetos da imagem como [18,23]. Em outras, os vértices representam pixels [1]. Esta última abordagem foi utilizada neste trabalho.

Neste contexto, os arcos do grafo representam uma relação de adjacência entre os pixels. Esta relação de adjacência é binária e irreflexa. Na maioria das aplicações de processamento de imagens a relação de adjacência é *invariante à translação*, significando que ela depende apenas da posição relativa entre os pixels relacionados.

Um exemplo de uma relação de adjacência $\mathcal{A}$ consiste em todos os pares de pixels distintos (s,t) pertencentes à imagem $\mathcal{I}$, tal que $d(s,t) \leq \rho$, onde $d(s,t)$ denota a distância Euclideana entre os pixels s e t, e ρ é uma constante específica. As Figuras 3.1(a)-(c) mostram os pixels adjacentes a um pixel central na relação de adjacência Euclideana com

$\rho = 1$ (4-conexa), $\rho = \sqrt{2}$ (8-conexa) e $\rho = \sqrt{5}$ respectivamente.

Observe, porém, que o conceito de adjacência depende da topologia do grafo e não da imagem. De uma forma genérica, uma relação assimétrica poderia ser definida considerando os pares de pixels (s,t) onde $t - s \in \mathcal{M}$, $s \neq t$, e $\mathcal{M}$ é um subconjunto de Z^2. Por exemplo, a Figura 3.1(d) ilustra os adjacentes de um pixel s quando $\mathcal{M} = \{(-1,-1),(1,-1)\}$.

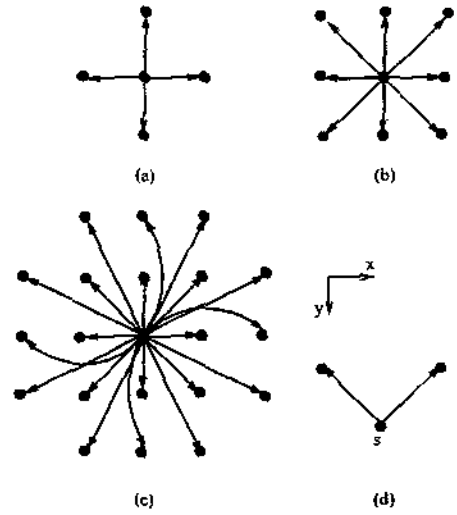


Figura 3.1: (a)-(c) Relação de adjacência Euclideana para $\rho = 1, \sqrt{2}$, e $\sqrt{5}$, respectivamente. (d) Relação de adjacência assimétrica.

3.2 Caminhos, conexidade e particionamento de imagens

Um *caminho* π no grafo é uma seqüência de pixels distintos $\langle t_1, t_2, \ldots, t_k \rangle$ onde $(t_i, t_{i+1}) \in A$ para $1 \leq i \leq k-1$. Um caminho é dito *trivial* se $k = 1$. Se π é um caminho que termina em um pixel t e τ é um caminho que inicia em t, $\pi \cdot \tau$ é o caminho resultante da concatenação de π e τ, com as duas instâncias de t unidas em uma.

Um pixel s é dito conexo a outro pixel t se existe um caminho π de s a t no grafo que representa a imagem. Um conjunto de componentes conexos e disjuntos cuja união não totaliza a imagem, representa uma partição da imagem.

Deste modo, seja um problema no qual deseja-se obter uma partição da imagem, onde cada componente conexo recebe um rótulo $l = 1, 2, 3, ..., k$ e o número k depende

da relação de adjacência $\mathcal{A}$. Podemos dizer que dois pixels $(s,t) \in \mathcal{A}$, se $d(s,t) \leq \rho$ e $\delta(s,t) = |I(s) - I(t)| \leq T$, onde $I(s)$ representa o brilho do pixel s na imagem original e T é um limiar de dissimilaridade. Aplicando um filtro com esta adjacência e $\rho = 2$ sobre a imagem da Figura 3.2(a) obtem-se cada letra, o fundo, cada pontuação e cada acento como uma partição como mostra a Figura 3.2(b). O mesmo filtro com $\rho = 5$ separa as palavras e fundo em partições distintas como mostra a Figura 3.2(c). Por fim, aplicando um filtro com adjacência retangular, ao invés da distância Euclideana, com largura de 30 pixels e altura de 5 pixels consegue-se separar cada linha e fundo em partições distintas conforme a Figura 3.2(d).

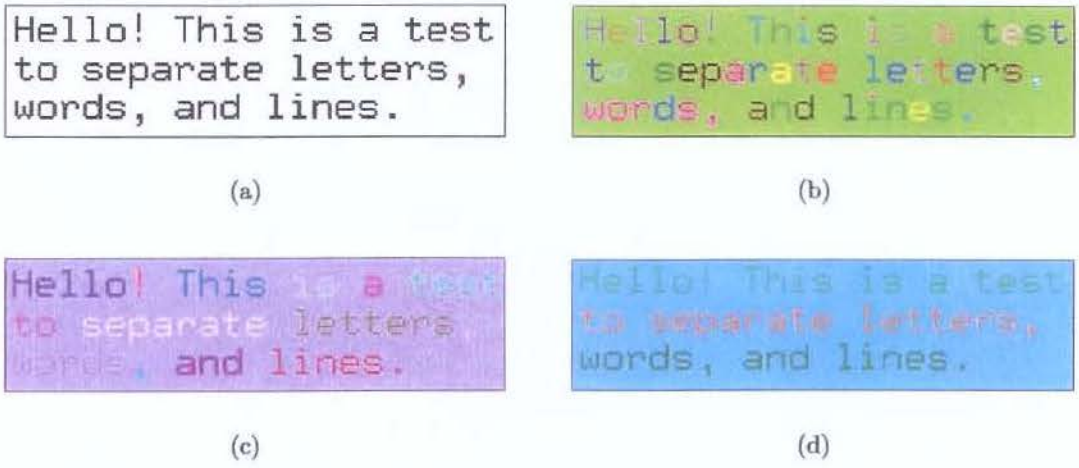


Figura 3.2: Problema de particionamento da imagem em forma de um grafo. (a) Imagem original contendo uma frase em três linhas. (b) Filtro de distância Euclideana de raio $\rho = 2$ para particionar letras, acentos, pontuações e fundo (c) Filtro de distância Euclideana de raio $\rho = 5$ para particionar palavras e fundo. (d) Filtro retangular com largura = 30 pixels e altura = 5 pixels para particionar linhas e fundo.

3.3 Custos, funções de custo e distância entre pixels

O custo de um caminho π é determinado por uma *função de custo de caminho* $f(\pi)$, a qual normalmente depende de propriedades locais da imagem ao longo do caminho, tais como cor, gradiente e posição de pixel. Os possíveis valores de custo gerados por esta função encontram-se em um intervalo $\mathcal{V}$.

Uma função f é dita *monotomicamente incremental* quando atende aos seguintes requisitos:

$$f(\langle t \rangle) = h(t), \tag{3.1}$$
$$f(\pi \cdot \langle s,t \rangle) = f(\pi) \odot (s,t), \tag{3.2}$$

onde $h(t)$ é o custo do caminho trivial e a operação $\odot$ satisfaz as seguintes condições: $x' \geq x \Rightarrow x' \odot (s,t) \geq x \odot (s,t)$ e $x \odot (s,t) \geq x$ para x e $x' \in \mathcal{V}$ e $(s,t) \in \mathcal{A}$

As funções f_{sum} e $f_{\max}$, descritas abaixo, são os exemplos mais utilizados de funções monotonicamente incrementais.

$$f_{sum}(\langle t \rangle) = h(t), \tag{3.3}$$
$$f_{sum}(\pi \cdot \langle s,t \rangle) = f_{sum}(\pi) + w(s,t), \tag{3.4}$$

onde $w(s,t)$ é o peso fixo não negativo da aresta $\langle s,t \rangle$ e $h(t)$ é um custo inicial para o caminho $\langle t \rangle$.

$$f_{\max}(\langle t \rangle) = h(t), \tag{3.5}$$
$$f_{\max}(\pi \cdot \langle s,t \rangle) = \max\{f_{\max}(\pi), w(s,t)\}, \tag{3.6}$$

onde $w(s,t)$ é o peso fixo da aresta $\langle s,t \rangle$.

Um caminho π é dito *ótimo* se $f(\pi) \leq f(\pi')$ para qualquer outro caminho π' que termine no mesmo pixel t, independentemente do pixel inicial.

Cada pixel possui uma *distância* dele até todos os outros pixels. Caso um pixel t não seja conexo a outro s, então a distância de t a s é máxima, denotada por $+\infty$. Caso contrário a distância de t a s assume um valor $k \leq +\infty$.

A distância do pixel t ao pixel s, pode ser relacionada ao custo de caminho mínimo de t a s. Escolhendo-se uma função de custo de caminho, atribui-se um custo inicial $h(t)$ para o pixel t. Seja π o caminho ótimo de t para s, o custo do caminho π representa a distância de t para s.

Esta distância, medida por um custo de caminho mínimo, permite gerar um determinado particionamento da imagem dependendo da função de custo empregada.

Um exemplo típico de particionamento por meio da função de custo é o de separar alguns objetos do fundo de uma imagem. Para isso, são escolhidos um conjunto de pixels $o_1, o_2, ..., o_k$ como sementes para cada objeto $1, 2, ..., k$ e outro conjunto de pixels s como sementes para o fundo. A partir destas sementes são medidas distâncias para todos

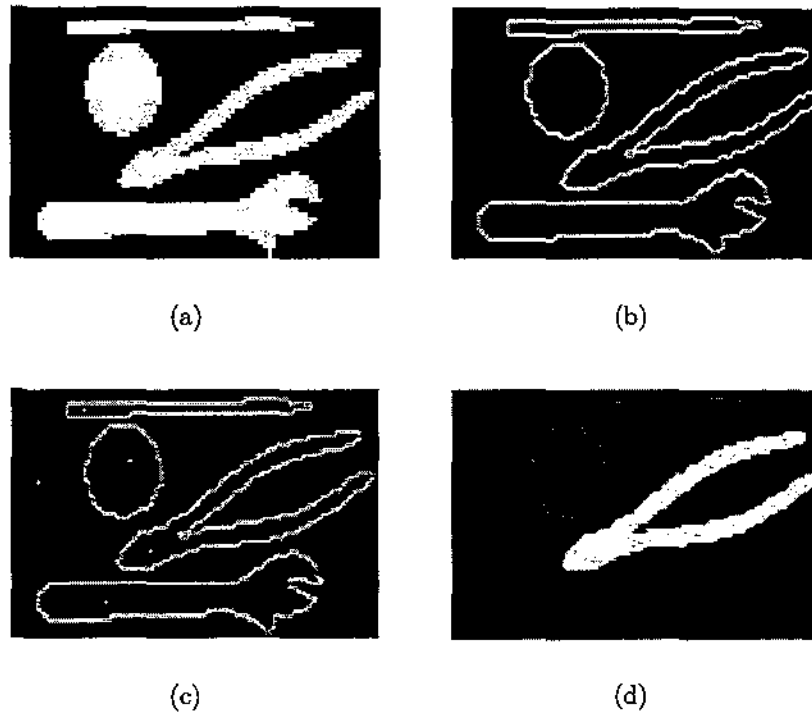


Figura 3.3: Particionamento de imagem em fundo e objetos de interesse calculando distâncias por meio de uma função de custo de caminho. (a) Imagem inicial. (b) Gradiente da imagem inicial (c) Imagem de gradiente com sementes utilizadas. (d) Imagem particionada em quatro objetos e fundo.

os outros pixels da imagem. Define-se então uma partição de cada objeto pelos pixels mais próximos das sementes em $o_1, o_2, ..., o_k$ e uma partição de fundo com os pixels mais próximos das sementes em s. A Figura 3.3 demonstra um particionamento de 4 objetos e o fundo da imagem. Utilizou-se o gradiente da imagem inicial com um pixel semente interno a cada objeto e um no fundo. A adjacência utilizada foi 8-conexa e a função de custo de caminho foi:

$$\begin{aligned} f_{\max}(\langle t \rangle) &= 0, t \in \mathcal{S}, \qquad (3.7) \\ &= +\infty \; cc., \\ f_{\max}(\pi \cdot \langle s,t \rangle) &= max\{f_{\max}(\pi), I(t)\}, \qquad (3.8) \end{aligned}$$

Onde, $\mathcal{S}$ é o conjunto de todas as sementes e $I(t)$ é o brilho do pixel t na imagem de gradiente.

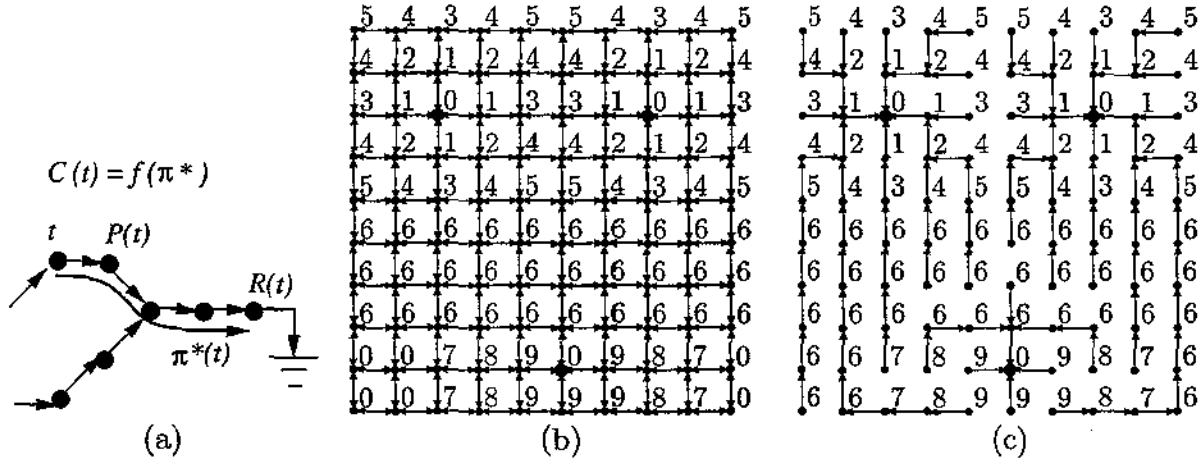


Figura 3.4: (a) Os principais elementos de uma floresta. (b) Uma imagem como um grafo orientado com adjacência-4. (c) Uma floresta de caminhos ótimos.

3.4 Mapa de predecessores e florestas em imagens

Um *mapa de predecessores* é uma função P que associa a cada pixel t na imagem $\mathcal{I}$ ou algum outro pixel em $\mathcal{I}$, ou uma marca *nil* que não pertence a $\mathcal{I}$ — neste caso t é dito ser uma *raiz* do mapa. Uma *floresta* é um mapa de predecessores que não possui ciclos — isto é, aquela função que leva todos os pixels a *nil* em um número finito de avaliações. Para qualquer pixel $t \in \mathcal{I}$, uma floresta F define um caminho $\pi^*(t)$ recursivamente como $\langle t \rangle$ se $P(t) = nil$, e $\pi^*(s) \cdot \langle s, t \rangle$ se $P(t) = s \neq nil$, como ilustrado pela Figura 3.4(a), onde $R(t)$ é o pixel inicial ou raiz de $\pi^*(t)$. Assim como se vê nas Figuras 3.4(b)-(c), uma *floresta de caminhos de custo mínimo* é uma floresta F onde $\pi^*(t)$ é ótimo para todo pixel $t \in \mathcal{I}$.

Não só o problema de separar objetos e fundo, como visto na Seção anterior, pode ser resolvido explorando estes conceitos. Outros problemas tais como delinear o segmento ótimo de contorno de um objeto, encontrar distância geodésica entre conjuntos de pixels e realizar transformadas de distância, entre outros, também podem ser tratados.

Na verdade, explorando o conceito de floresta de caminhos de custo mínimo, uma vasta gama de problemas pode ser resolvido. Este conceito é exatamente o mecanismo base da IFT.

Capítulo 4

Transformada Imagem-Floresta

A IFT é uma ferramenta genérica utilizada para gerar, implementar e validar operadores de processamento de imagem baseados em conexidade. A partir de uma função de custo de caminho e de uma imagem representada como um grafo, tendo cada pixel um custo de caminho mínimo inicial, algumas sementes e uma relação de adjacência, a IFT produz uma floresta de caminhos de custo mínimo.

Neste capítulo serão abordadas tanto questões do funcionamento quanto das aplicações da IFT em processamento de imagens. O algoritmo básico e algumas restrições de funções de custo de caminho e conjuntos de sementes para a IFT encontram-se na Seção 4.1. Na Seção 4.2 apresentam-se algumas das aplicações da IFT em processamento de imagens. Na Seção 4.3 estão descritas duas possíveis implementações da IFT. Na Seção 4.4 descreve-se a implementação diferencial da IFT. A Seção 4.5 conclui o capítulo com uma motivação para a implementação da IFT em hardware.

4.1 Algoritmo e restrições da IFT

O algoritmo básico da IFT é o seguinte:

Entrada: Uma imagem $\mathbf{I} = (\mathcal{I}, I)$; uma relação de adjacência $\mathcal{A} \subset \mathcal{I} \times \mathcal{I}$; e uma função de custo de caminho f.

Saída: Uma floresta de caminho ótimo F.

Estruturas de dados auxiliares: Dois conjuntos de pixels $\mathcal{F}$ e $\mathcal{Q}$ cuja união é $\mathcal{I}$.

1. Definir $\mathcal{F} = \{\}$, $\mathcal{Q} \leftarrow \mathcal{I}$. Para cada $t \in \mathcal{I}$ faça $P(t) \leftarrow nil$.
2. Enquanto $\mathcal{Q} \neq \{\}$ faça
   2.1. Remover de $\mathcal{Q}$ um pixel s tal que $f(P^*(s))$ é mínimo e adicionar s em $\mathcal{F}$.
   2.2. Para cada pixel t, tal que $(s,t) \in \mathcal{A}$ faça

2.2.1. Se $f(P^*(s) \cdot \langle s,t \rangle) < f(P^*(t))$ faça $P(t) \leftarrow s$.

Onde $P^*(t)$ é uma definição recursiva de P, na qual $P^*(t) = \langle t \rangle$ se $P(t) = nil$ e $P^*(t) = P^*(s) \cdot \langle s,t \rangle$ se $P(t) = s \neq nil$.

O algoritmo tem complexidade $O(n)$ no número de iterações referente ao laço externo e $O(m)$ referente ao laço interno. Isto ocorre, pois em cada iteração do laço externo, exatamente um pixel é removido do conjunto $\mathcal{Q}$, nunca retornando e todas as arestas do pixel analisado são analisadas exatamente uma vez, pelas iterações do laço interno.

Este algoritmo pode ser reestruturado para otimizar os cálculos de custo e para gerar além do mapa de predecessores da floresta de caminhos de custo mínimo, um mapa dos custos de cada caminho e outro das raízes da floresta. Os mapas de custos e de raiz são matrizes com as dimensões da imagem processada. No mapa de custos, cada elemento (x, y) da matriz possui o custo de caminho mínimo até o pixel daquela posição. Já no mapa de raízes, cada elemento (x, y) possui o rótulo correspondente à raiz da árvore à qual o pixel daquela posição pertence. A Figura 4.1 contém um exemplo com (a) a imagem de entrada e (b) os três mapas de saída. Estas alterações da IFT estão pormenorizadas nas implementações descritas na Seção 4.3

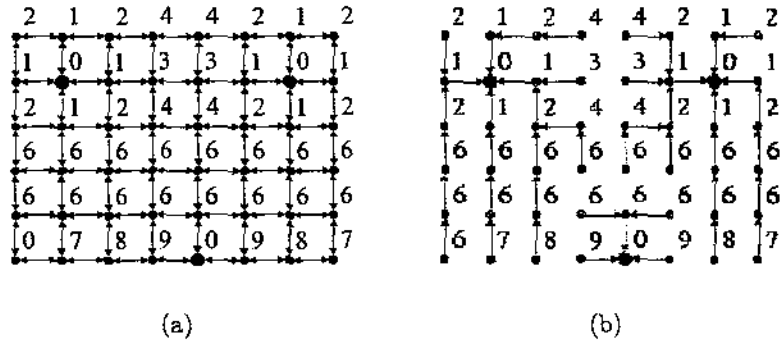


Figura 4.1: Dados de saída da IFT. (a) Imagem de entrada como grafo com adjacência representada por arestas e pixels por vértices, com o valor do seu brilho. Os pontos grandes na imagem são os vértices sementes. (b) Grafo representando os mapas de saída da IFT, utilizando adjacência 4-conexa e função de custo $f_{\max}$. Vértices representam pixels com o mapa de custo na floresta de caminhos de custo mínimo, arestas representam mapa de predecessores e cor representa o mapa de raízes.

Na seção anterior, foi visto o conceito de funções monotonicamente incrementais com os exemplos de $f_{\max}$ e f_{sum}. O algoritmo da IFT funciona corretamente para este tipo de função. No entanto, as funções de custo de caminho que geram uma floresta ótima fazem

parte de um conjunto maior de funções, observando-se porém, que nem toda função de custo de caminho produz um resultado ótimo.

Seja um conjunto de sementes $\mathcal{S} = \{r_1, r_2\}$ e a função de custo f_{abs} definida como:

$$f_{abs}(\langle t \rangle) = h(t), \tag{4.1}$$
$$f_{abs}(\pi \cdot \langle s, t \rangle) = |I(t) - I(R(s))|, \tag{4.2}$$

onde, $I(t)$ corresponde ao brilho do pixel t, $R(s)$ é a raiz do pixel s e $h(t)$ é 0 caso t seja uma semente e $+\infty$ caso contrário.

Seja uma imagem de 3x2 pixels, com seus respectivos brilhos iniciais, ilustrada na Figura 4.2. Ao aplicar-se o algoritmo da IFT para esta imagem, com a função de caminho f_{abs} e a relação de adjacência 4-conexa, o resultado final é incorreto. Neste caso o grafo de caminhos ótimos pode não ser uma floresta, mas um grafo com ciclos.

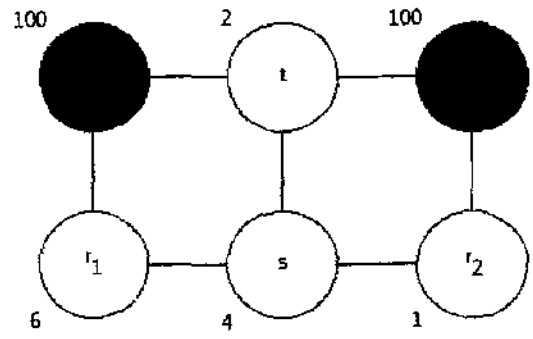


Figura 4.2: Imagem de 3x2 pixels, na qual a IFT falha ao computar o custo, utilizando a função de custo de caminho f_{abs}.

Durante a IFT, o pixel s é conquistado pelo pixel r_1, pois o menor custo de caminho oferecido provém dele sendo de $|4 - 6| = 2$. Posteriormente, o menor custo de caminho oferecido para t é o proveniente de s. Como s foi conquistada por r_1, o custo oferecido a t é dado por $|I(t) - I(r_1)| = |2 - 6| = 4$. No entanto, este custo não é ótimo, já que se o caminho tivesse origem em r_2, o custo oferecido de s para t seria $|I(t) - I(r_2)| = |2 - 1| = 1$. Portanto, para a função de custo f_{abs} com vizinhança 4-conexa, o algoritmo da IFT falha em produzir a floresta de caminhos de custo mínimo.

Como foi dito, a função de custo de caminho utilizada pela IFT não precisa ser necessariamente monotonicamente incremental para gerar uma floresta ótima. Por exemplo, a

função de custo que propaga a distância Euclideana a partir das sementes do grafo não é monotonicamente incremental. Para um conjunto de sementes $\mathcal{S}$ com um ou dois pixels o algoritmo da IFT produz um resultado correto com esta função. Já para a mesma função de custo de caminho, com três ou mais sementes, o resultado da IFT pode não ser correto.

Um conceito mais abrangente para a função pela qual a IFT produz uma floresta de caminhos de custo mínimo é definido como *função suave*, tendo as seguintes características: Para todo pixel $t \in \tau$ existe um caminho ótimo π terminando em t que, ou é trivial, ou tem a forma $\tau \cdot \langle s, t \rangle$, onde

1. $f(\tau) \leq f(\pi)$;
2. τ é um caminho ótimo e;
3. para qualquer caminho τ' terminado em s, $f(\tau' \cdot \langle s, t \rangle) = f(\pi)$.

Uma função monotonicamente incremental é também suave. No entanto, como visto para a função de distância Euclideana, a suavidade da função depende também do conjunto de sementes e da adjacência escolhidos. Conforme é visto na Figura 4.3, utilizando-se três sementes e a relação de adjacência 8-conexa, o valor produzido pela IFT pode ser não ótimo.

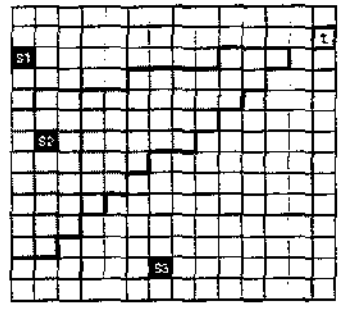


Figura 4.3: Imagem com distâncias Euclideanas com origem nas sementes s_1, s_2 e s_3. Neste caso, não há uma floresta de caminhos mínimos produzida pela IFT com adjacência 8-conexa e função de custo de caminho baseada na distância Euclideana. O pixel t é conquistado com um custo de caminho maior que o mínimo. A função não é suave.

Utilizando diferentes funções de custo de caminhos suaves, combinadas com conjuntos distintos de sementes e adjacências variadas, muitos operadores de processamento de imagens podem ser implementados e, conseqüentemente, abre-se espaço para diversas aplicações para a IFT.

4.2 Aplicações da IFT

Como visto na Seção 4.1, a saída da IFT é a floresta de caminhos de custo mínimo, representada pelo mapa de predecessores. Pode-se gerar também um mapa de custos e um de raízes da floresta correspondente (Figura 4.1), com pequenas alterações do algoritmo. A IFT reduz, com isso, uma grande variedade de operadores de processamento de imagens a um processamento local simples sobre estes mapas.

A Figura 4.4 exemplifica a solução para o operador de caminho geodésico mínimo entre dois conjuntos de pixels. Foi utilizada uma imagem arbitrária com um conjunto de pixels A, que forma um corredor claro ligando os dois conjuntos de pixels S e D, cinzas claro e escuro respectivamente, dentro de um conjunto de pixels B que forma um fundo preto. As sementes da IFT são os pixels do conjunto cinza claro. A adjacência utilizada é vizinhança-8 com função de custo de caminho dada por:

$$\begin{aligned} f_{Chamf}(\langle t \rangle) &= 0 \rightarrow t \in S, \\ &= -\infty \rightarrow t \in B, \\ &= +\infty \rightarrow t \in A \cup D, \end{aligned} \tag{4.3}$$

$$\begin{aligned} f_{Chamf}(\pi \cdot \langle s,t \rangle) &= f_{Chamf}(\pi) + 5 \rightarrow d(s,t) = 1, \\ &= f_{Chamf}(\pi) + 7 \rightarrow d(s,t) = \sqrt{2}, \end{aligned} \tag{4.4}$$

onde $d(s,t)$ é a distância Euclideana entre s e t, t recebe $-\infty$ se pertence ao fundo preto para não ser conquistado e o caminho resultante não passar por ele e t recebe $+\infty$ se pertencer ao corredor claro ou conjunto de pixels de destino, cinza escuro. O custo de caminho oferecido durante o percurso sempre aumenta quanto mais longe o pixel se encontra das sementes. O pixel do conjunto de destino que possuir menor custo de caminho representará o último pixel do caminho geodésico mínimo entre os dois conjuntos.

Outro exemplo clássico de operador tratado pela IFT é o pipeline para esqueletonização. Entende-se por esqueleto um lugar geométrico dos centros dos discos de raios máximos contidos em um contorno. Os discos tocam o contorno em mais de um ponto, mas sem cruzarem-no. Esqueletos podem ser utilizados como representações compactas de objetos, úteis para reconstruir, filtrar detalhes, registrar e extrair características dos mesmos.

O operador descrito a seguir extrai o esqueleto multi-escala de um objeto contido em uma imagem. Ele utiliza-se de duas IFT's, sendo a primeira para gerar uma imagem de rótulos de objeto e fundo e a segunda para propagar rótulos. Posteriormente, calcula-se uma imagem de diferença e por fim o esqueleto.

(a) (b)

Figura 4.4: (a) Imagem arbritária. (b) Caminho geodésico mínimo encontrado entre componentes cinzas da imagem arbitrária

A primeira IFT aplicada é uma IFT-watershed. Ela consiste de uma IFT sobre o gradiente morfológico da imagem de entrada com um conjunto de sementes interno ao objeto e outro conjunto de sementes de fundo, com função de custo de caminho:

$$f_{\max}(\langle t \rangle) = I(t) + 1, \tag{4.5}$$

$$f_{\max}(\pi \cdot \langle s,t \rangle) = \max\{I(t), f_{\max}(\pi)\}, \tag{4.6}$$

onde $I(t)$ é o brilho do pixel t, sendo $I(t) < +\infty$ para todo t e utilizando uma adjacência 8-conexa. As sementes internas ao objeto recebem um rótulo de mesmo valor o e todas as sementes de fundo recebem rótulo de valor b. Cada partição do objeto e do fundo deve conter pelo menos uma semente. Obtem-se como resultado a segmentação do objeto do fundo, tendo os pixels de objeto rótulo o e os pixels de fundo rótulo b, na floresta gerada. A segmentação de células de neurônios é ilustrada na Figura 4.5(a). Note que o objeto pode conter mais de uma partição e pode conter também furos, pertencentes ao fundo. Como conseqüência, o objeto pode conter também diversos contornos desconexos, que o separam do fundo.

A partir do mapa de rótulos da floresta segmentada, os pixels de contorno entre o objeto e o fundo são detectados. Para isto, basta checar os pixels que contém rótulo o e pelo menos um pixel adjacente de rótulo b, em uma adjacência 4-conexa. Rotulam-se então cada um destes contornos por um valor inteiro consecutivo, ou seja, cada pixel do primeiro contorno recebe o rótulo um, do segundo rótulo dois e assim por diante, em uma ordem arbitrária dos contornos. Dentro de cada contorno, os pixels recebem outro

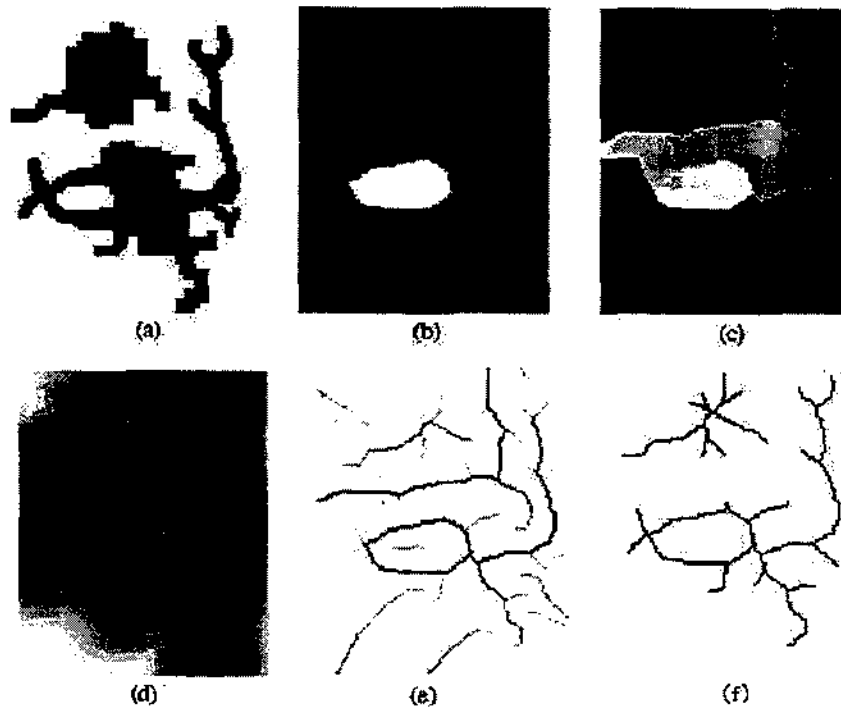


Figura 4.5: Imagem que ilustra esqueletonização multi-escala de neurônios.(a) Imagem dos rótilos o e b, referentes ao objeto e ao fundo respectivamente, após segmentação. (b)-(d) rótulo de contorno, rótulo de pixel em contorno e distâncias Euclideanas propagadas pela IFT. (e) Imagem Diferença (f) esqueleto do objeto procurado, ou seja, os neurônios.

rótulo seqüencialmente por inteiros crescentes, enquanto o contorno é percorrido pelo seu formato externo.

Aplica-se então uma IFT-transformada de Chamfer para propagar estes rótulos pela imagem. Esta IFT utiliza a imagem inicial como entrada. O conjunto de sementes é formado pelos pixels de contorno do objeto, com os dois rótulos atribuídos. A função de custo de caminho utilizada é:

$$f_{Chamf}(\langle t \rangle) = 0 \rightarrow t \in \mathcal{S}, \tag{4.7}$$

$$= +\infty \; cc,$$

$$f_{Chamf}(\pi \cdot \langle s,t \rangle) = f_{Chamf}(\pi) + 5 \rightarrow d(s,t) = 1, \tag{4.8}$$

$$= f_{Chamf}(\pi) + 7 \rightarrow d(s,t) = \sqrt{2},$$

onde S é o conjunto de todas as sementes e $d(s,t)$ é a distância Euclideana entre s e t, com adjacência 8-conexa. Como retorno obtem-se o mapa dos rótulos de contorno L_c, o mapa dos rótulos dos pixels de cada contorno L_p propagados e o mapa das distâncias de Chanfer. As Figuras 4.5(b)-(d) evidenciam rotulos de contorno e de pixel e as distâncias Euclideanas propagadas da imagem dos neurônios.

A partir destes mapas de rótulos, uma imagem de diferença D é definida. Para isso, duas imagens de diferença intermediárias D_1 e D_2 são computadas da seguinte forma:

$$D_1 \leftarrow \max_{\forall t \in N_4(s)} \{L_c(t) - L_c(s)\},$$

$$D_2 \leftarrow \max_{\forall t \in N_4(s)} \{L_p(t) - L_p(s)\},$$

onde $N_4(s)$ representa o cunjunto de pixels 4-conexo a s. Se $D_1(s) > 0$, então $D_1(s) \leftarrow M$, onde M é o rótulo máximo atribuido a um pixel em L_s, senão $D_1(s) \leftarrow 0$. Se $D_2(s) > N/2$, onde N é o número de pixels no contorno de label $L_c(s)$, então $D_2(s) \leftarrow N - D_2(s)$.

Depois disso, a imagem de diferença D é calculada por:

$$D(s) \leftarrow \max\{D_1(s), D_2(s)\}$$

Esta imagem de diferença contém o esqueleto multiescala tanto do objeto quanto do fundo da imagem como mostra a Figura 4.5(e). Basta escolher então o esqueleto do fundo por selecionar todo pixel p com o valor $D(s) > 0$, com rótulo o do objeto. Figura 4.5(f) contém o esqueleto dos neurônios.

Mais um exemplo de aplicação que utiliza a IFT é a segmentação de objetos por contorno de borda. O método consiste em percorrer segmentos consecutivos do contorno do objeto com o uso da IFT até que seja completada uma volta por toda a sua borda.

Primeiramente gera-se o gradiente oposto da imagem de entrada. As Figuras 4.6(a)-(b) contém a imagem original e de gradiente oposto de um cerebelo. A imagem gradiente oposto é gerada da imagem gradiente por inverter o brilho dos pixels. Deste modo, os pixels com gradiente maior possuem brilho menor na imagem gradiente oposto. A seguir seleciona-se pixels ordenados, s_0, s_1, s_2, ..., s_n, pertencentes à borda do objeto que se deseja segmentar.

Processa-se então uma IFT com s_0 como sementes, adjacência 8-conexa e a função de custo:

$$\begin{aligned} f_{sum}(\langle t \rangle) &= 0 \rightarrow t \in \mathcal{S}, \\ &= +\infty \ cc., \end{aligned} \tag{4.9}$$

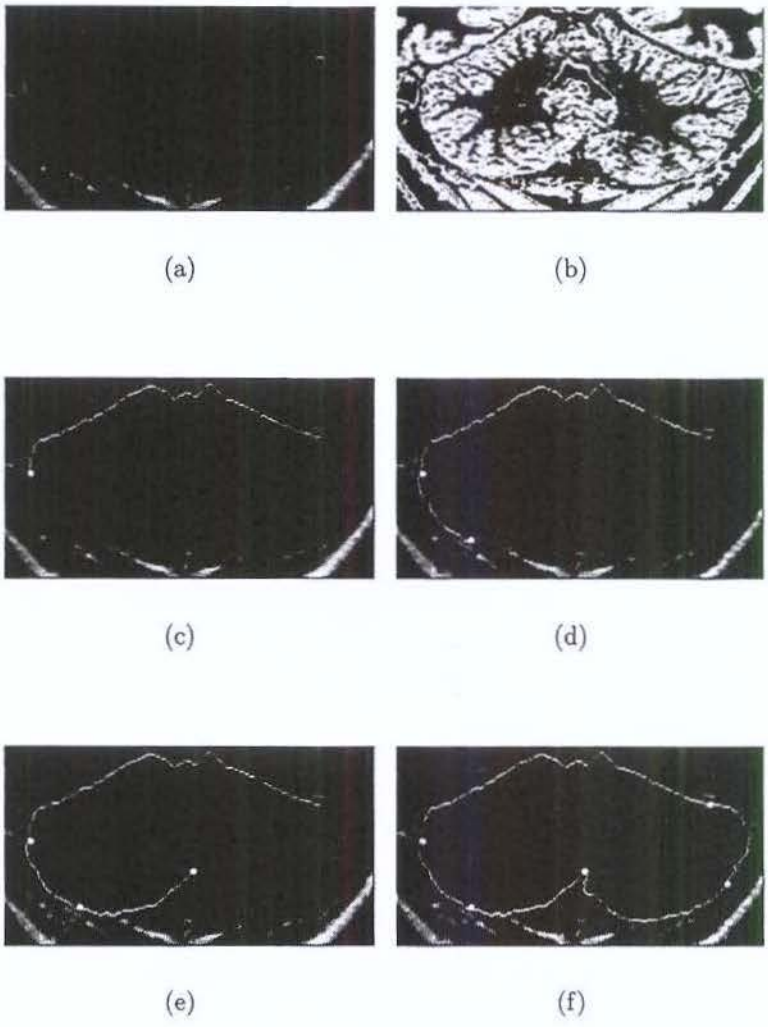


Figura 4.6: Contorno de borda de um objeto utilizando a IFT. (a) Imagem de um cerebelo. (b) Gradiente oposto calculado a partir da imagem (a). (c)-(e) Primeiro ao terceiro segmentos percorridos pela IFT. (f) Contorno completo do cerebelo.

$$f_{sum}(\pi \cdot \langle s,t \rangle) = f_{sum}(\pi) + I(t), \tag{4.10}$$

onde S é o conjunto de todas as sementes e I(t) é o brilho da imagem gradiente oposto no pixel t. A Figura 4.6(c) contém este primeiro passo na segmentação baseando-se em uma escolha aleatória de sementes sobre o contorno do objeto.

Uma nova IFT é processada em seqüência na mesma imagem gradiente oposto, com a semente s_1, a mesma adjacência e a seguinte função de custo:

$$\begin{aligned} f_{sum}(\langle t \rangle) &= C(t) => t \in \langle s_0, s_1 \rangle, & (4.11) \\ &= +\infty \; cc, \\ f_{sum}(\pi \cdot \langle s,t \rangle) &= f_{sum}(\pi) + I(t), & (4.12) \end{aligned}$$

onde $C(t)$ é o custo final da IFT anterior para o pixel t e $\langle s_0, s_1 \rangle$ é o caminho de s_0 a s_1 calculado pela IFT anterior, sendo o mapa de predecessores mantido para este caminho.

Esta aplicação mantém o custo do melhor caminho de s_0 a s_1 e permite que os outros pixels da imagem sejam reconquistados. A Figura 4.6(d) mostra a segunda aplicação da IFT.

Este procedimento é repetido com s_2 no conjunto de sementes, mantendo os custos e predecessores do caminho de s_0 a s_2. O terceiro passo do contorno é mostrado na Figura 4.6(e). O mesmo se repete para todos os pixels semente até s_n. Um detalhe para a última iteração é que os valores $C(t)$ são mantidos para todo caminho de s_0 a s_n, excluindo-se o próprio pixel s_0, para que possa ser reconquistado. Figura 4.6(f) ilustra o contorno final do cerebelo.

Ao final do processo, uma volta com o caminho de s_0 a s_0 no mapa de predecessores define o contorno da borda do objeto desejado. Esta aplicação é muito apropriada, pois a borda do objeto é preferivelmente escolhida por causa de seus menores custos. Além disso, por utilizar-se uma função de custo de caminho de adições de custo, permite-se que existam pequenas descontinuidades ao longo da borda, sem que o percurso desvie-se das proximidades do objeto.

Vários outros operadores podem ser implementados pela IFT por ser um algoritmo genérico. Dentre eles pode-se citar transformada de Watershed, reconstruções morfológicas [7]), transformada de distância Euclideana e operadores relacionados [8–10], filtragem, segmentação e análise de imagens [2,3,11], reconstrução morfológica e operadores relacionados [7], dimensão fractal multi-escala e pontos de saliência de curvas [10] e perseguição de bordas [13–15].

4.3 Filas de prioridade e implementações

O gargalo do algoritmo da IFT apresentado na Seção 4.1 encontra-se no passo 2.1, a seleção do pixel de menor custo de caminho $s \in \mathcal{Q}$. Se $\mathcal{Q}$ fosse implementada como uma estrutura de heap, o tempo do algoritmo seria de $O(m + n\ log\ n)$.

Na maioria das aplicações, as funções de custo de caminho realizam incrementos de custos inteiros limitados a uma constante K ao longo do caminho. Isto permite a utilização de uma fila circular com $K + 1$ posições. Cada posição i, $i = 0, 1, ..., K$ deve armazenar uma lista duplamente ligada de todo pixel t com custo i igual ao resto da divisão $C(t)/K$. Como o tamanho máximo do grafo $|\mathcal{I}|$ que representa a imagem é conhecido, estas listas duplamente ligadas podem ser implementadas em uma única matriz A de ponteiros $A.next(t)$ e $A.prev(t)$ com $\mathcal{I}$ elementos. Neste caso, o algoritmo da IFT terá complexidade $O(m + nK)$. A fila de prioridade circular e suas filas duplamente ligadas internas estão ilustradas na Figura 4.7. Caso o valor de K não seja conhecido de antemão para alguma aplicação, pode-se utilizar uma abordagem flexível K', na qual o tamanho da fila cresce caso a diferença do maior ao menor elemento da fila seja maior que K'.

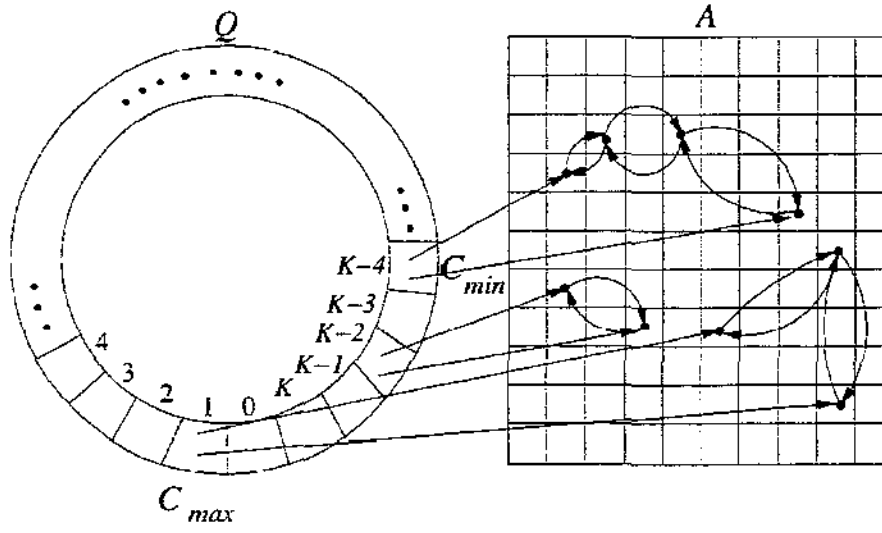


Figura 4.7: Fila de prioridade Circular $\mathcal{Q}$, com $K + 1$ elementos. O elemento C_{min} possui o menor valor dentre os pixels da fila e o C_{max} o de maior valor. Cada elemento da fila é uma lista duplamente ligada de apontadores para uma matriz de elementos A de tamanho $|\mathcal{I}|$.

Esta fila de prioridade circular naturalmente ordena por custos o processamento dos

pixels. No entanto, dentro de cada lista duplamente ligada, todos os pixels têm o mesmo custo. Sendo assim, necessita-se de uma política de desempate para a seleção de algum deles. Normalmente, a política mais utilizada e a FIFO (*First In First Out*). Em alguns casos, porém, a política LIFO (*Last In First Out*) pode ser mais precisa. A partir de cada uma delas formulou-se uma implementação otimizada do algoritmo da IFT.

O algoritmo da IFT com política de desempate FIFO é o seguinte:

Entrada: Uma imagem $\mathbf{I} = (\mathcal{I}, I)$; uma relação de adjacência $\mathcal{A} \subset \mathcal{I} \times \mathcal{I}$; e uma função de custo de caminho f.

Saída: Imagem com mapa de custos $\mathbf{C} = (\mathcal{I}, C)$, imagem com mapa de predecessores $\mathbf{P} = (\mathcal{I}, P)$ e imagem com mapa de raízes $\mathbf{R} = (\mathcal{I}, R)$.

Auxiliares: Fila de prioridade $\mathcal{Q}$ e variável tmp.

1. Para todo $t \in \mathcal{I}$ faça:
   1.1. $C(t) \leftarrow f(\langle t \rangle)$, $P(t) \leftarrow nil$ e $R(t) \leftarrow t$.
   1.2. Se $C(t) < +\infty$, insira t em $\mathcal{Q}$.
2. Enquanto $\mathcal{Q} \neq \emptyset$ faça:
   2.1. Remova um pixel t de $\mathcal{Q}$, cujo custo $C(t)$ é mínimo.
   2.2. Para todo $s \in A(t)$, tal que $C(s) > C(t)$, faça:
      2.2.1. $tmp \leftarrow f(P^*(t) \cdot \langle t, s \rangle)$.
      2.2.2. Se $tmp < C(s)$ faça:
         2.2.2.1. Se $C(s) \neq +\infty$, remova s de $\mathcal{Q}$.
         2.2.2.2. $C(s) \leftarrow tmp$, $P(s) \leftarrow t$, $R(s) \leftarrow R(t)$ e insira s em $\mathcal{Q}$.

O algoritmo da IFT com política de desempate LIFO é o seguinte:

Entrada: Uma imagem $\mathbf{I} = (\mathcal{I}, I)$; uma relação de adjacência $\mathcal{A} \subset \mathcal{I} \times \mathcal{I}$; e uma função de custo de caminho f.

Saída: Imagem com mapa de custos $\mathbf{C} = (\mathcal{I}, C)$, imagem dom mapa de predecessores $\mathbf{P} = (\mathcal{I}, P)$ e imagem com mapa de raízes $\mathbf{R} = (\mathcal{I}, R)$.

Auxiliares: Fila de prioridade $\mathcal{Q}$ e variável tmp.

1. Para todo $t \in \mathcal{I}$ faça:
   1.1. $C(t) \leftarrow f(\langle t \rangle)$, $P(t) \leftarrow nil$, $R(t) \leftarrow t$ e insira t em $\mathcal{Q}$.
2. Enquanto $\mathcal{Q} \neq \emptyset$ faça:
   2.1. Remova um pixel t de $\mathcal{Q}$, cujo custo $C(t)$ é mínimo.
   2.2. Para todo $s \in A(t)$, tal que $s \in \mathcal{Q}$, faça:
      2.2.1. $tmp \leftarrow f(P^*(t) \cdot \langle t, s \rangle)$.

2.2.2. Se $tmp \leq C(s)$ faça:
2.2.2.1. Remova s de $\mathcal{Q}$, faça $C(s) \leftarrow tmp$, $P(s) \leftarrow t$, $R(s) \leftarrow R(t)$ e insira s em $\mathcal{Q}$.

4.4 Algoritmo seqüencial diferencial

Em algumas operações de processamento de imagens, tais como segmentação de objetos, é interessante calcular seqüências de IFT's. Em uma seqüência, cada IFT utiliza os mapas de custo, predecessor e raiz gerados pela IFT anterior. Utiliza-se também a mesma função de custo de caminho e adjacência, para conjuntos subseqüentes de sementes S_p, $p = 1, 2, ..., n$. Talvez se deseje ainda que, a cada instante p entre as IFT's, que se removam árvores marcadas por pixels em conjuntos M_p, $p = 1, 2, ..., n$ dos mapas gerados.

Podemos imaginar então um processo de segmentação onde, a cada instante p, os conjuntos S_p e M_p são informados e a IFT é calculada de forma diferencial com relação à floresta no instante $p-1$. As sementes S_p conquistam regiões de pixels mais conexas com elas do que com as raízes da floresta no instante $p-1$. As árvores com pixels no conjunto M_p serão removidas e suas regiões disponibilizadas para novas conquistas, por parte das raízes restantes e das novas sementes contidas em S_p. Estas raízes são representadas por pixels das árvores não removidas que são adjacentes aos pixels das árvores removidas. Estes pixels são denominados *pixels de fronteira*. O processo pára quando a floresta satisfaz a segmentação desejada.

O algoritmo da IFT diferencial (DIFT) é o seguinte:

Entrada: Uma imagem $\mathbf{I} = (\mathcal{I}, I)$, uma relação de adjacência $\mathcal{A} \subset \mathcal{I} \times \mathcal{I}$, uma função de custo de caminho f, imagem com mapa de custos $\mathbf{C} = (\mathcal{I}, C)$, imagem com mapa de predecessores $\mathbf{P} = (\mathcal{I}, P)$, imagem com mapa de raízes $\mathbf{R} = (\mathcal{I}, R)$, e os conjuntos S_p e M_p. Na primeira iteração, $C(t) = +\infty$ e $P(t) = nil$ para todo $t \in \mathcal{I}$.

Saída: Imagem com mapa de custos $\mathbf{C} = (\mathcal{I}, C)$, imagem com mapa de predecessores $\mathbf{P} = (\mathcal{I}, P)$ e imagem com mapa de raízes $\mathbf{R} = (\mathcal{I}, R)$ no instante p.

Auxiliares: Fila de prioridade $\mathcal{Q}$ com política FIFO, variável tmp, e conjunto F de pixels de fronteira.

1. Faça $(\mathbf{C}, \mathbf{P}, F) \leftarrow RemoveArvores(\mathbf{C}, \mathbf{PR}, A, M_p)$.
2. Enquanto $F \neq \emptyset$ faça:
   2.1. Remova um pixel s de F.
   2.2. Se $s \notin S_p$ insira s em $\mathcal{Q}$.
3. Enquanto $S_p \neq \emptyset$ faça:
   3.1. Remova um pixel s de S_p.
   3.2. Se $f(\langle s \rangle) < C(s)$, então $C(s) \leftarrow f(\langle s \rangle)$, $P(s) \leftarrow nil$, $R(s) \leftarrow s$, e insira s em $\mathcal{Q}$.

4. Enquanto $\mathcal{Q} \neq \emptyset$ faça:
   4.1. Remova um pixel t de $\mathcal{Q}$, cujo custo $C(t)$ é mínimo.
   4.2. Para todo $s \in A(t)$, tal que $C(s) > C(t)$, faça:
      4.2.1. $tmp \leftarrow f(P^*(t) \cdot \langle t, s \rangle)$.
      4.2.2. Se $tmp < C(s)$ ou $P(s) = t$ faça:
         4.2.2.1. Se $s \in \mathcal{Q}$, remova s de $\mathcal{Q}$.
         4.2.2.2. $C(s) \leftarrow tmp$, $P(s) \leftarrow t$, $R(s) \leftarrow R(t)$ e insira s em $\mathcal{Q}$.

O algoritmo da *RemoveArvores* é o seguinte:

Entrada: Imagem com mapa de custos $\mathbf{C} = (\mathcal{I}, C)$, imagem com mapa de predecessores $\mathbf{P} = (\mathcal{I}, P)$, imagem com mapa de raízes $\mathbf{R} = (\mathcal{I}, R)$, adjacência $\mathcal{A}$ e o conjunto M_p.

Saída: Imagem com mapa de custos $\mathbf{C} = (\mathcal{I}, C)$ e imagem com mapa de predecessores $\mathbf{P} = (\mathcal{I}, P)$ atualizados após remoção de árvores e conjunto F de pixels de fronteira.

Auxiliares: Fila de prioridade T com política FIFO, conjunto M de raízes marcadas para remoção e variável r.

1. Enquanto $M_p \neq \emptyset$ faça:
   1.1. Remova um pixel s de M_p.
   1.2. Faça $r \leftarrow R(s)$
   1.2. Se $C(r) \neq +\infty$, então insira r em T e em M e faça $C(r) \leftarrow +\infty$ e $P(r) \leftarrow nil$.
2. Enquanto $T \neq \emptyset$ faça:
   2.1. Remova um pixel t de T.
   2.2. Para todo $s \in A(t)$ faça:
      2.2.1. Se $P(s) = t$ então $C(s) \leftarrow +\infty$ e $P(s) \leftarrow nil$ e insira s em T.
      2.2.2. Se não, mas se $R(s) \notin M$ então insira s em F.

O algoritmo *RemoveArvore* reinicializa os mapas C e P, percorrendo em largura cada árvore removida da floresta e retorna em F os pixels de fronteira. A principal diferença entre o algoritmo da DIFT e o algoritmo original é o teste do predecessor, $P(s) = t$ na linha 4.2.2. Sem este teste, o mapa de raízes pode ficar desatualizado. Isto é, se algum pixel, cuja raiz manteve-se na IFT anterior, for alcançado por uma nova semente com o mesmo custo, ele deve ser reconquistado, caso seu predecessor também tiver sido reconquistado. Um exemplo do processo de reconquista de regiões consta na Figura 4.8.

Este algoritmo da IFT tem como principal vantagem a redução de tempo de processamento de seqüências de IFT's que diferem apenas no conjunto de sementes nos dados de entrada. Um exemplo da aplicação da DIFT encontra-se na Figura 4.9.

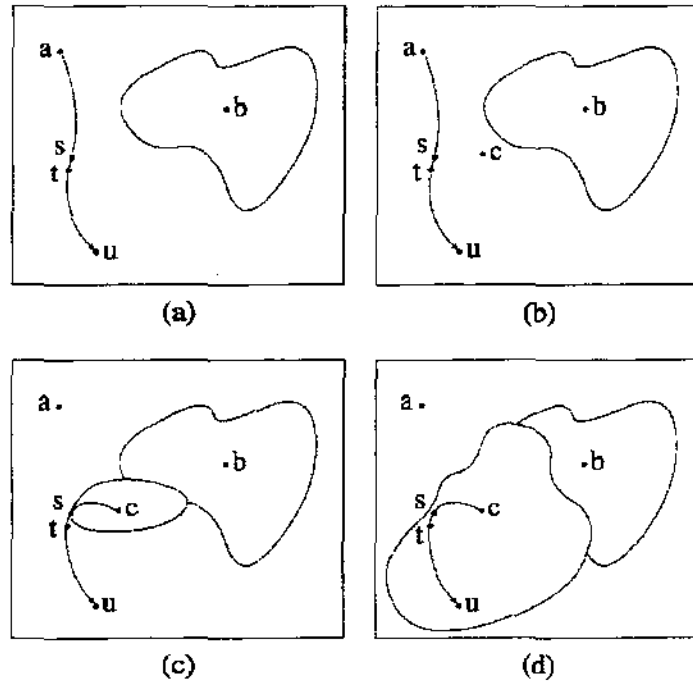


Figura 4.8: Reconquista de pixels com mesmo predecessor. (a) Caminho ótimo em floresta com duas árvores com raízes nos pixels em a e b, onde o caminho $\langle a, ..., s, t, ..., u\rangle$ é ótimo. (b) Semente c acrescentada. (c) Durante a DIFT $seqc, ..., s$ é encontrado, sendo mais barato que $\langle a, ..., s\rangle$. Mas $C(t) = f(seqc, ..., s, t)$. O teste de predecessor garante que todo pixel cujo caminho ótimo passa por s, como por exemplo o sufixo $\langle t, ..., u\rangle$, será reconquistado e receberá o rótulo da raiz c.(d) Um possível resultado desta DIFT.

4.5 Conclusão

A IFT mostra-se ser uma ferramenta muito útil e extremamente rápida para diversas operações de processamento de imagens. Para imagens pequenas como 200x150 pixels, uma IFT é processada em um PC moderno em cerca de 10ms.

Entretanto, algumas aplicações, que requerem um resultado imediato por percepção humana ou por um programa executado em um PC, demandam muito tempo de processamento.

Para processar uma seqüência de operações sobre uma ou mais imagens, ou até mesmo para imagens grandes, o tempo de processamento pode ser de alguns segundos, minutos ou mais [3].

Neste contexto, surge a *SIFT*, uma implementação em hardware da IFT, que permite

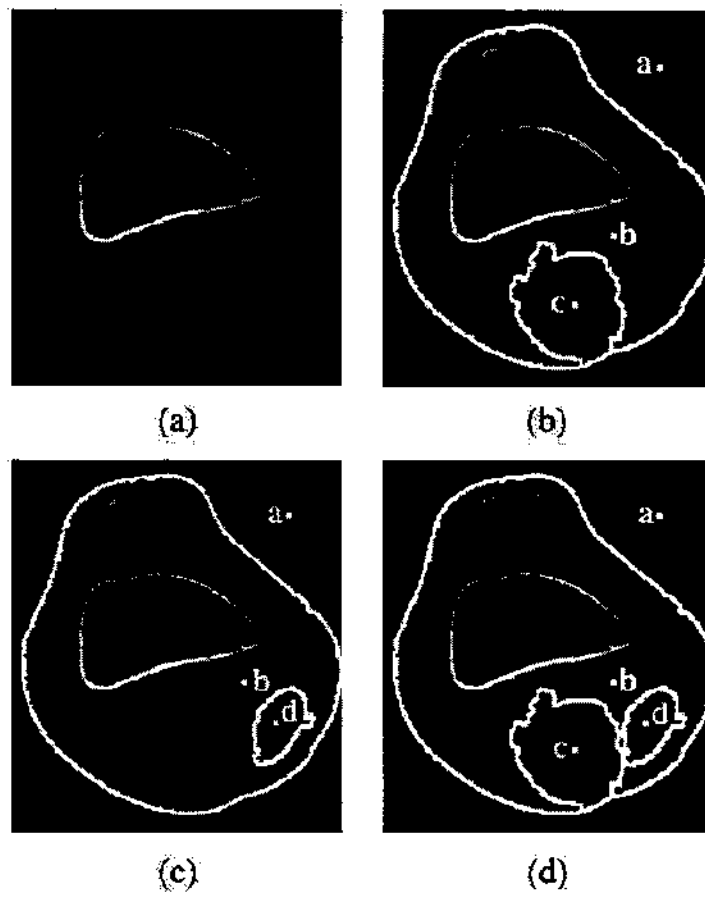


Figura 4.9: Um exemplo de segmentação de imagem utilizando a DIFT. (a) Imagem de um joelho (b) Floresta inicial com três raízes, $\{a, b, c\}$. (c) Estado depois de computação diferencial com adição da semente d e remoção de c(DIFT com $S_2 = \{d\}$ e $M_2 = \{c\}$). (d) Após re-adição de c(DIFT com $S_3 = \{c\}$ e $M_3 = \emptyset$).

uma maior velocidade de processamento.

Capítulo 5

SIFT

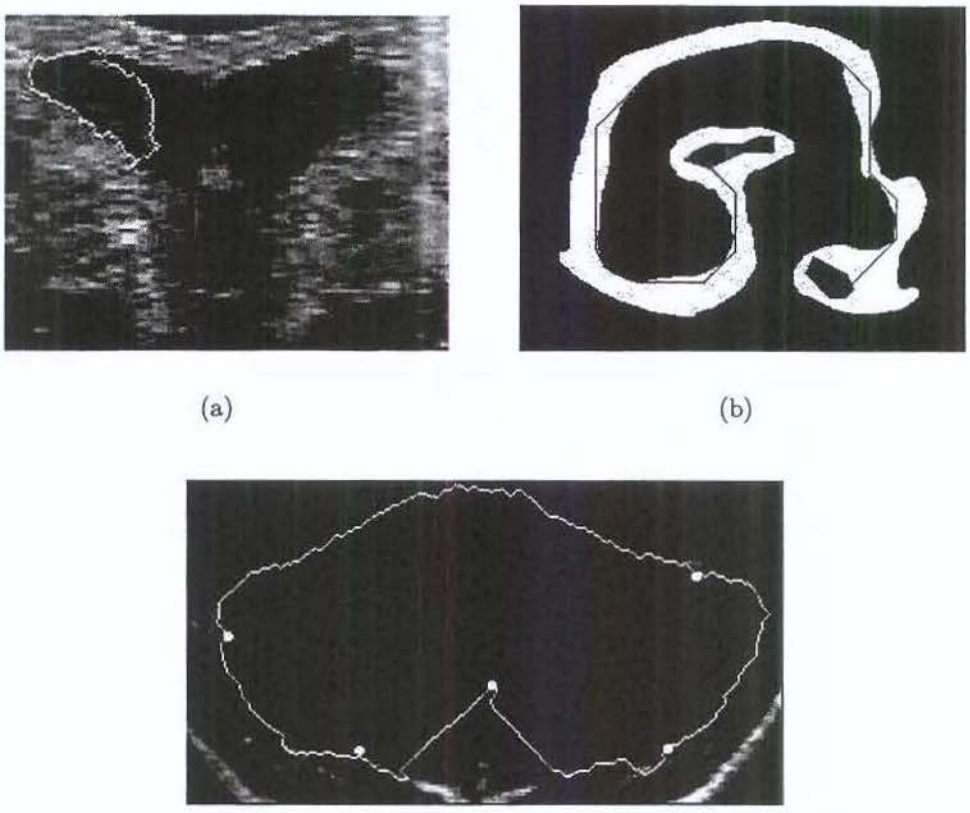
A SIFT é uma plataforma em *hardware* que contém uma implementação da IFT [1]. O hardware utilizado é uma FPGA, pois exige menor tempo de projeto e recursos financeiros, para o desenvolvimento, quando comparado a outras arquiteturas. Por ser uma plataforma em hardware, aproveita-se também de paralelismo de imagem no seu processamento, como visto na Seção 2.

Entrando-se com uma imagem de brilho, uma imagem de custo, as sementes iniciais e uma função de custo de caminho, a SIFT produz como resultado um mapa de predecessores que contêm uma floresta de caminho mínimo e um mapa de custos com os valores dos custos de caminho mínimo referentes a esta floresta.

Diferentemente da IFT em *software*, a SIFT não produz um mapa de raízes ou rótulos da floresta de caminhos mínimos. Este rótulo seria dispendioso para ser calculado em hardware e pode ser determinado através do mapa de predecessores em tempo linear, se necessário.

Os operadores implementados pela SIFT são um subconjunto dos executados pela IFT em software. A SIFT é capaz de segmentar objetos por crescimento de regiões, segmentar objetos por contorno de bordas, encontrar caminhos mínimos entre conjuntos de pixels por regiões especificadas e processar outros operadores que se baseiem nas funções de custo e vizinhanças dela. Veja os resultados das simulações da SIFT na Figura 5.1.

Outras questões relacionadas ao tempo de processamento e tamanho da FPGA tiveram que ser relevadas para uma implementação correta e eficaz da plataforma. A adjacência implementada na SIFT está restrita a apenas duas das possíveis pela IFT. Pode-se escolher a vizinhança 4-conexa e a vizinhança 8-conexa que são respectivamente, as vizinhanças de circunferências de raios 1.0 e 1.5. Elas atendem a requisitos de roteamento de dados entre processadores como será descrito na próxima Seção. As funções de custo da SIFT são $f_{\max}$ e f_{sum}. Estas funções geram uma unidade aritmética relativamente pequena, sendo possível instanciar um grande número delas na plataforma.



(a) (b)

(c)

Figura 5.1: Operações de processamento de imagem realizadas pela SIFT. (a) Segmentação do núcleo caudado do cérebro por crescimento de região. (b) Caminho geodésico encontrado a partir do conjunto cinza escuro até o conjunto cinza claro. (c) Segmentação do cerebelo por contorno de borda.

Inicialmente, um modelo global, sem levar em conta limitações físicas da FPGA foi proposto. Este modelo é simples, oferecendo uma visão mais clara do problema e pode ser implementado em futuras FPGA's maiores ou em outras arquiteturas de hardware, como por exemplo um ASIC.

Posteriormente, um modelo alternativo, menor em área de implementação, satisfez muito bem os requisitos desejados de maneira simples e elegante, com um tempo de processamento ainda muito abaixo da IFT em software. Ambos os modelos são detalhados a seguir.

5.1 Modelo Global

A idéia deste modelo é a de criar uma plataforma que possua uma unidade de processamento para cada pixel da imagem. O algoritmo seqüencial da IFT descrito na Seção 4 teve que ser adaptado para tirar vantagem do paralelismo das unidades de processamento.

A Fila de prioridade deixa de ser atraente, pois isto limita o poder de processamento de hardware. Além disso, a fila de prioridade circular, com uma lista duplamente ligada em cada posição, como utilizada na IFT em software, é uma estrutura muito complexa para o hardware, tanto em termos de implementação como quanto em termos de funcionamento, como mostra a Figura 4.7.

Por outro lado, a fila de prioridade garante a corretude do algoritmo e o seu término depois de processar no máximo todos os pixels uma vez.

Um algoritmo paralelo, sem filas de prioridade, foi projetado para que a SIFT possa gerar resultados corretos e parar em algum instante do processamento.

Para compreender o algoritmo é necessária uma visualização da estrutura e comportamento dos componentes contidos na SIFT.

5.1.1 Estrutura da SIFT no Modelo Global

A unidade de processamento para cada pixel da imagem, na SIFT, é denominada *IFT Processor - (IFP)*. A IFP é composta de uma interface de comunicação com IFP's de pixels adjacentes denominada *IFP Dada Manager - (IDM)*, uma unidade aritmética para o cálculo das funções de custo denominada *IFP Dada Processor - (IDP)*, um controlador de estados internos, para controlar a transmissão de custos para IFP's vizinhas *IFP Monitor - (IM)* e uma memória interna que guarda o custo, o brilho, um bit semente e o predecessor do pixel que representa *IFP Registers - (IR)*. O bit semente indica se o pixel é semente do grafo. Toda esta estrutura da IFP está evidente na Figura 5.2.

Além de uma matriz de IFP's com as dimensões da imagem, a SIFT possui uma unidade chamada *SIFT Controller - (SC)*. A SC coordena a entrada e saída dos dados da SIFT. Ela também controla as IFP's, antes durante e depois do processamento. A Figura 5.3 mostra a SIFT de um panorama global.

A SC está ligada a cada IFP por alguns barramentos de controle e dois barramentos de dados. O controle da IFP é composto de:

- uma entrada de 3 bit chamada *we*. Este barramento indica que um dado externo de brilho, custo, bit semente ou predecessor deve ser gravado na memória IR. A SC possui um barramento *we* para cada linha de IFP's, ou pixels da imagem de entrada. Assim, a matriz de IFP's recebe dados para a SC linha por linha.

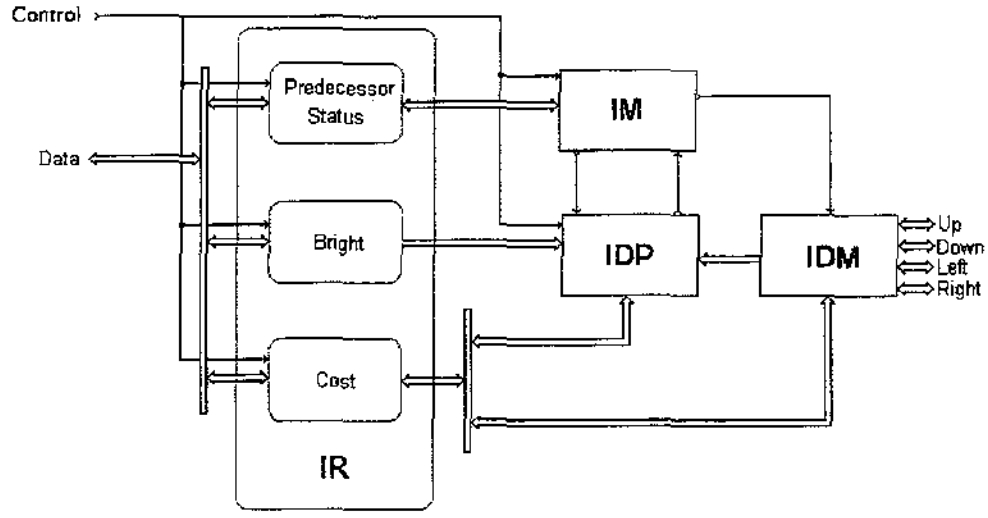


Figura 5.2: Estrutura e componentes da IFP.

- uma entrada de 2 bits chamada *re*. Este barramento indica qual dado interno de custo, bit semente ou predecessor deve ser gravado na memória IR. A SC possui um barramento *re* para todas as IFP's ou pixels da imagem de entrada.

- uma entrada de 1 bit chamada *run*. Este barramento indica que a IFP deve realizar o processamento quando ativado. Quando desativado indica que a IFP está disponível para leitura e escrita de dados pela SC. A SC tem um único barramento *run* ligado a todas as IFP's

- uma entrada de 3 bits chamada *state*. Este barramento é utilizado durante o processamento e controla o roteamento dos dados entre as IFP's. Apenas um barramento *state* sai da SC para todas as IFP's.

- uma saída de 1 bit chamada *ready*. Este barramento indica para a SC quando IFP termina seu processamento. Há um barramento *ready* saindo de cada IFP para a SC.

Os dados entram na IFP por um barramento de entrada de 8 bits e saem por um sinal de 8 bits chamados *data_in* e *data_out* respectivamente. Existe um barramento de entrada e outro de saída da SC para cada coluna de IFP's. A escolha da largura desses barramentos foi baseada na largura de entrada da interface externa à SIFT e na largura dos dados de custo, brilho, predecessor e bit semente para a saída. Veja os sinais e roteamento descrito nas Figuras 5.4 e 5.5

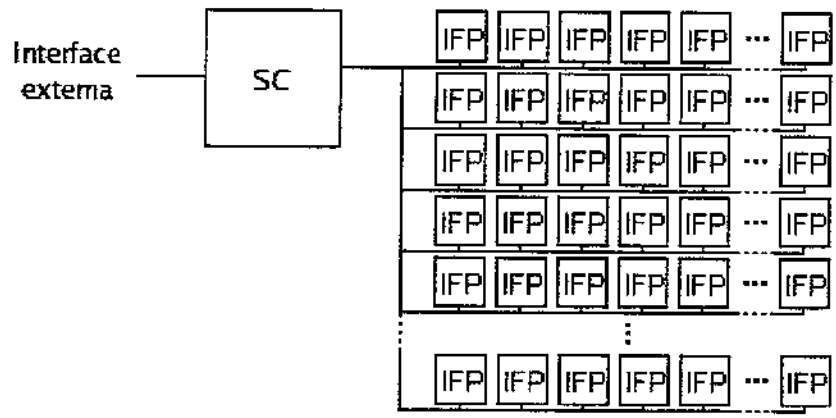


Figura 5.3: Estrutura e componentes da SIFT.

Além dos barramentos entre a SC e as IFP's, as IFP's estão ligadas entre si em uma vizinhança 4-conexa por barramentos bidirecionais de 16 bits, pelos quais realizam-se transferências de dados necessárias durante o processamento da imagem. A IDM (IFP Dada Manager) controla a entrada e saída dos dados nos barramentos, conforme o estado interno da IM, o estado externo da SC e a adjacência escolhida. Ela é capaz de direcionar os dados do barramento provenientes de uma IFP vizinha para outra, para que a comunicação entre vizinhos 8-conexos seja possível. Veja um exemplo de transmissão de dados nas vizinhanças 4 e 8-conexas durante uma fase na Figura 5.6

A unidade aritmética da IFP, IDP, implementa o cálculo das funções de custo de caminho. A IDP implementa as funções suaves $f_{\max}$ e f_{sum}, descritas no Capítulo 3.

Há uma limitação para os argumentos destas funções. Dado um pixel t propagando seu custo para um pixel adjacente s, a função de custo de caminho $f_{\max}$ retorna o máximo entre o custo c_t e o brilho b_s. Já para função f_{sum} pode-se escolher fazer a soma do custo c_t com um valor constante atribuído à relação de adjacência entre t e s ou fazer a soma do custo c_t com o brilho b_s. As funções de caminhos de custo mínimo implementadas são:

$$f_{\max}(\langle t \rangle) = h(t), \tag{5.1}$$
$$f_{\max}(\pi \cdot \langle t, s \rangle) = \max\{c_t, b_s\}, \tag{5.2}$$

$$f_{sum}(\langle t \rangle) = h(t), \tag{5.3}$$
$$f_{sum}(\pi \cdot \langle t, s \rangle) = c_t + b_s, \tag{5.4}$$

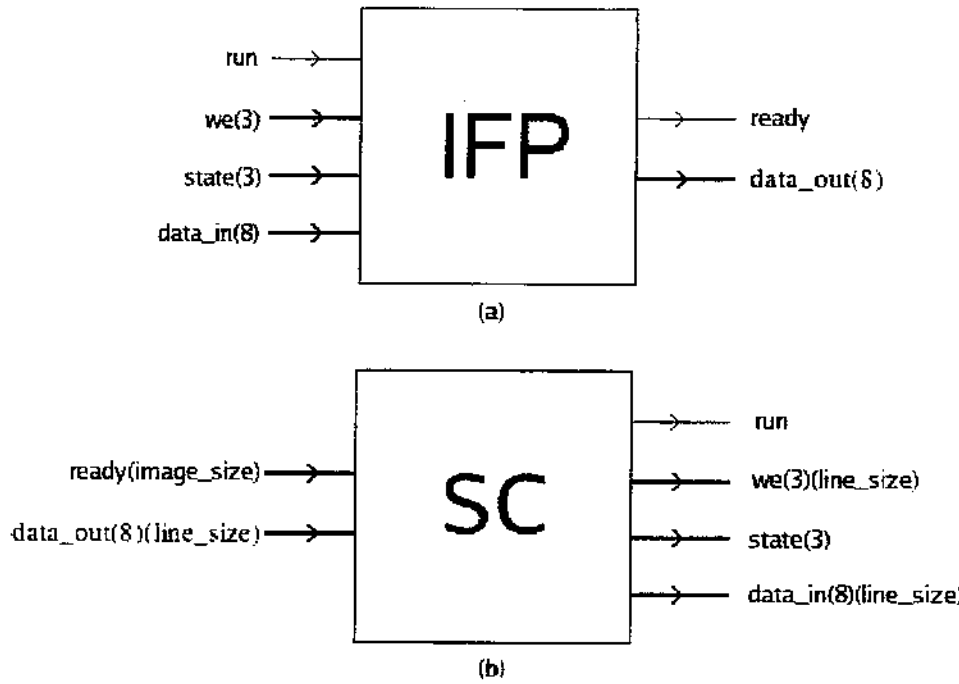


Figura 5.4: Roteamento de (a) uma IFP e (b) da SC com seus respectivos barramentos de entrada e saída. Os índices entre parênteses depois dos nomes dos barramentos indicam a largura dos mesmos, sendo de uma ou duas dimensões. Os barramentos representados por linhas finas são de apenas 1 bit.

$$
\begin{aligned}
f_{Chamf}(\langle t \rangle) &= 0 \rightarrow t \in \mathcal{S}, && (5.5)\\
&= +\infty \ cc, \\
f_{Chamf}(\pi \cdot \langle s,t \rangle) &= f_{Chamf}(\pi) + 5 \rightarrow d(s,t) = 1, && (5.6)\\
&= f_{Chamf}(\pi) + 7 \rightarrow d(s,t) = \sqrt{2},
\end{aligned}
$$

onde $\mathcal{S}$ é o conjunto de todas as sementes e $h(t)$ é um custo inicial, escolhido pelo usuário, atribuído ao pixel t.

O controlador de estados internos da IFP, IM, como é de se esperar, é dotado de uma máquina de estados. Ele indica para a IDM se os custos da IFP devem ou não ser transmitidas para as IFP's vizinhas, controla o predecessor da IFP na floresta de caminhos de custo mínimo e é responsável pelo barramento *ready* que indica para a SC o fim do processamento da sua parte. Seu funcionamento será descrito em detalhes na próxima Seção.

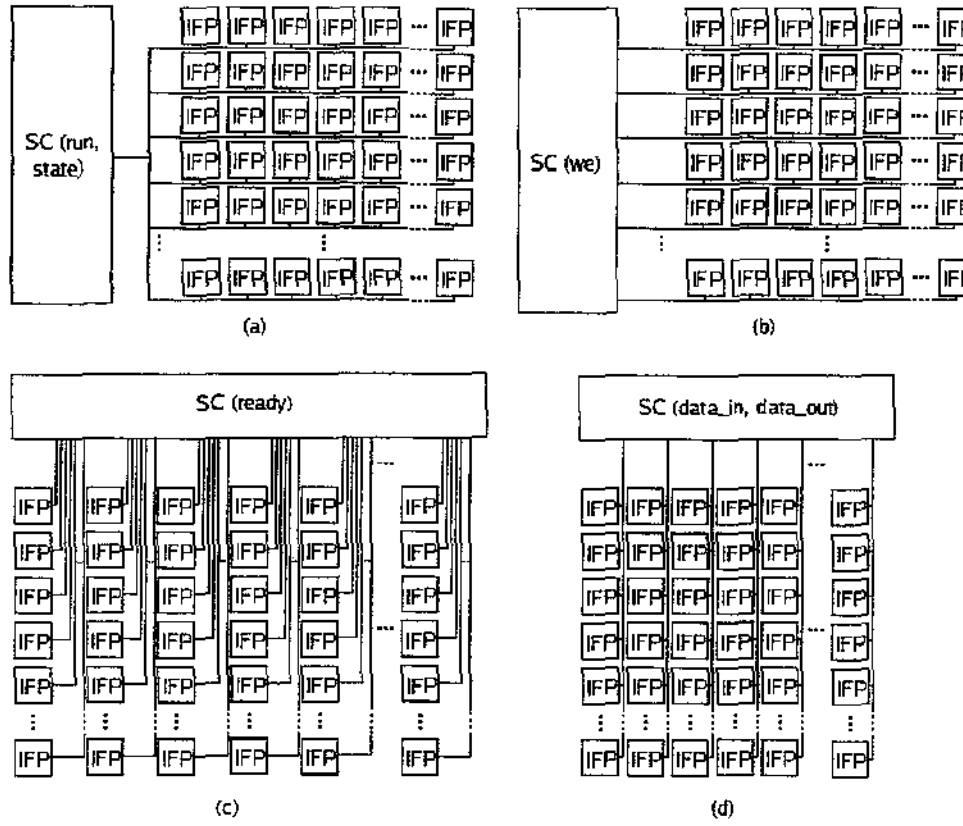


Figura 5.5: Barramentos entre a SC e a matriz de IFP's. Configuração dos barramentos (a) *run*, *state* e *re* (b) *we* (c) *data_in* e *data_out* (d) *ready*

A memória interna da IFP, IR, é constituída de um registrador de 37 bits que guarda os valores de brilho, custo, predecessor e bit semente referentes ao pixel. Os valores de brilho e custo são representados por números de 16 bits utilizando-se um registrador para cada, enquanto que o predecessor ocupa 4 bits e o bit semente apenas 1 bit em um único registrador.

5.1.2 Comportamento da SIFT no Modelo Global

Essencialmente, todo o processo realizado pela SIFT pode ser resumido em três etapas:

1. Entrada de dados da imagem;
2. Processamento da Imagem pelo algoritmo da IFT em paralelo;
3. Retorno dos resultados obtidos.

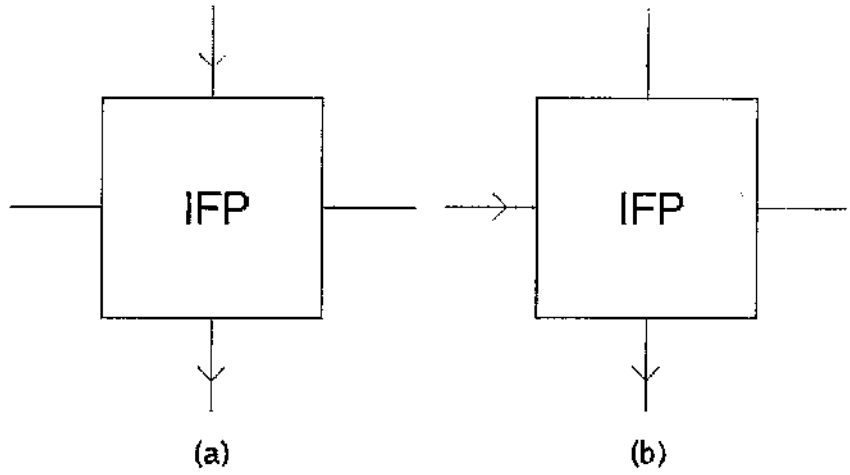


Figura 5.6: Transmissão de dados entre IFP's vizinhas. (a) Recepção de dados do vizinho superior e envio para o inferior em um típico processamento de vizinhança 4-conexa ou 8-conexa. (b) Recepção de dados do vizinho da esquerda, envio para o inferior e ligação do barramento do vizinho superior com o do vizinho da direita em um típico processamento de vizinhança 8-conexa.

A entrada de dados é realizada por uma seqüência ordenada de operações de leitura da interface externa controlada pela SC. Primeiramente, o brilho da imagem é lido da interface externa e transferido para as IFP's. Depois os custos de caminho iniciais são lidos e transferidos. Por último, são transferidos o bit semente dos pixels, a configuração inicial do mapa de predecessores da floresta e a vizinhança do pixel.

A leitura é controlada pela SC, que possui contadores de linha e coluna dos pixels da imagem, para entrar com os dados no endereço da IFP correta.

Durante o tempo de leitura o barramento *run* permanece com valor 0, o *state*, o *ready*, o *re* e o *data_out* não são levados em conta, o *data_in* transmite os dados e o *we* assume o valor '001' quando um brilho deve ser gravado, '010' quando um custo deve ser gravado e '100' quando o bit semente, o predecessor e a vizinhança devem ser gravados. A Figura 5.7 ilustra a temporização das operações[1].

No algoritmo da IFT em software, o mapa de predecessores inicial da floresta é descrito como um caminho trivial para todos os pixels da imagem [1]. No entanto, como visto no artigo da IFT diferencial [3], este mapa pode ser diferente do trivial e se ajustar corretamente à floresta mínima após o processamento.

[1]As formas de onda ilustradas nesta tese foram geradas pelo simulador do programa Quartus-II da Altera e os valores apresentados são todos em hexadecimal.

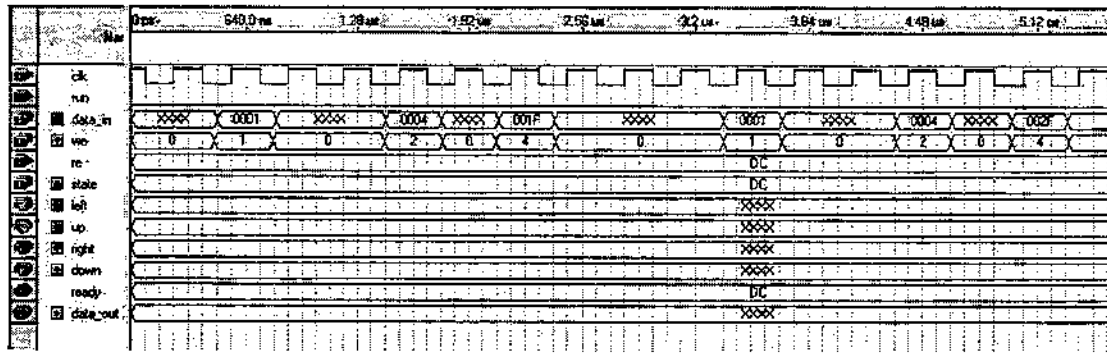


Figura 5.7: Comportamento dos sinais de entrada e saída de uma IFP durante a leitura de um valor de brilho, um valor de custo e um valor de predecessor, vizinhança e bit semente, nesta ordem.

No início do processamento, ao subir o bit do barramento *run*, as IFP's que possuem bit semente com valor 1, ou seja, são sementes, iniciam a propagação de seu custo de caminho para as IFP's vizinhas.

Durante todo o processamento, o barramento *run* assume o valor '1', os barramentos *data_out*, *we*, *re* e *data_in*, não são levados em conta. O barramento *state* informa as IFP's a fase atual e o barramento *ready* é monitorado pela SC, conforme a Figura 5.8. O processamento da SIFT é realizado em fases e ciclos de operação.

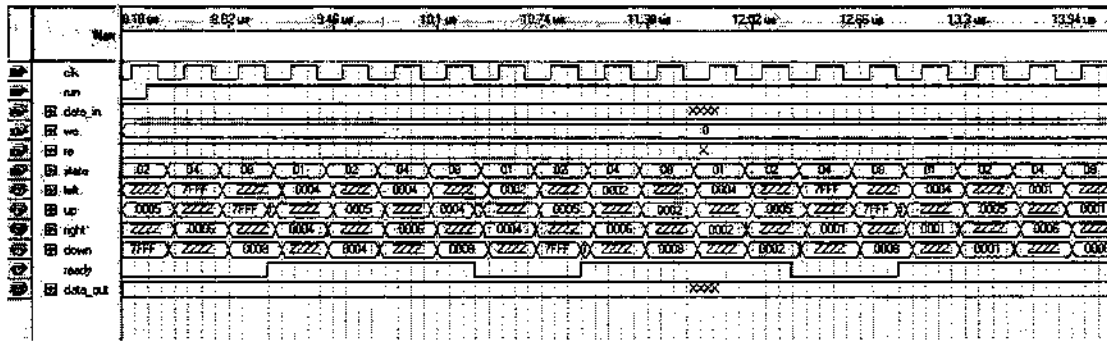


Figura 5.8: Comportamento dos sinais de entrada e saída de uma IFP durante o processamento da IFP. A IFP representa uma semente que começa seu processamento transmitindo seus dados aos vizinhos em um ciclo. Quando termina, um custo de caminho, de valor 2, proveniente do vizinho da esquerda, conquista a IFP. Ela faz a propagação do custo de caminho e logo em seguida é conquistada pelo vizinho da direita com custo de caminho 1. Este custo é então propagado e finalizam-as as ondas.

Uma fase tem a duração de um período do clock. Durante uma fase cada IFP pode propagar seu custo de caminho a uma das suas IFP's adjacentes. Também pode receber o

custo de caminho de outra de suas adjacentes, sendo ou não conquistada por um caminho de menor custo.

Um ciclo é composto por uma seqüência ordenada de fases, na qual uma IFP pode receber/enviar custos de caminho de/para todas IFP's adjacentes. A duração de um ciclo depende da relação de adjacência escolhida sendo de quatro fases para Vizinhança-4 e oito fases para Vizinhança-8.

A Figura 5.9 descreve as fases existentes em ambas as vizinhanças. Um ciclo passa por todas as fases da vizinhança, independentemente da fase inicial.

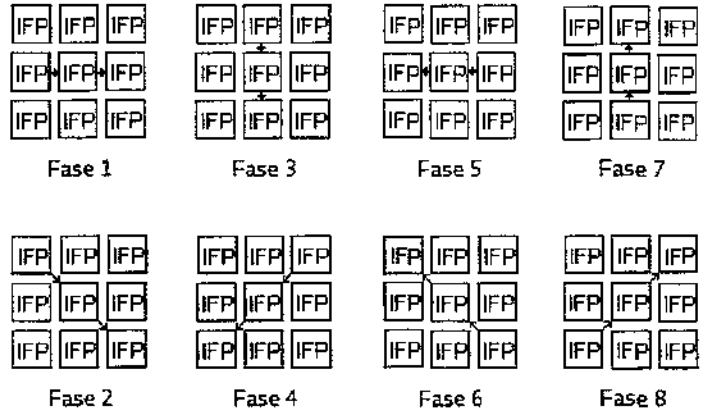


Figura 5.9: Fases da IFP. A propagação dos custos de caminho é indicada pelas setas em relação a IFP central. Quando operando em Vizinhança-4 o ciclo é composto apenas das fases ímpares. Quando operando em Vizinhança-8 o ciclo é composto de todas as fases ordenadamente.

Como visto anteriormente, a SIFT não é provida de uma fila de prioridade. Todo o pixel conquistado no algoritmo da IFT em software entrara numa fila e esperará a sua vez para propagar seus custos.

O comportamento paralelo da SIFT permite que todos os pixels, que precisam propagar os seus custos, comecem a fazê-lo instantaneamente, sejam sementes ou pixels conquistados. Além disso, todo o pixel pode ser conquistado um número indeterminado de vezes, sempre que chegue a ele um custo de caminho menor que o seu atual.

Evidentemente há um certo controle para que o mecanismo funcione e obtenha resultados corretos.

A SC coordena a operação das IFP's informando para elas a fase atual por meio do barramento *state*. Portanto, todas propagam os custos de caminho para a mesma adjacência. Desta forma, não há colisões de dados nos barramentos que ligam as IFP's adjacentes.

É importante notar que, para atingir uma IFP na adjacência diagonal, (fases pares na Figura 5.9), o custo de caminho é propagado primeiro na horizontal e depois na vertical. Desta forma não há colisões de dados nos barramentos.

A interface externa IDM da IFP é a responsável por fazer o roteamento dos barramentos, ligando a entrada e saída dos dados para a direção correta de propagação em cada fase.

Assim como o pixel tirado da fila de prioridade na IFT em software propagava seu custo de caminho a todos os adjacentes, o mesmo ocorre na SIFT com um pixel que propaga seu custo de caminho.

Por isso, cada IFP possui uma máquina de estados interna de um ciclo de fases, situado no IM, para controlar a propagação de seu custo de caminho. O IM conta internamente quantos pixels adjacentes restam para que o custo de caminho atual seja propagado em toda adjacência. Caso a IFP seja reconquistada, a contagem é reiniciada.

Durante a propagação de custos o bit do barramento *ready* da IFP permanece ligado e é desligado assim que o custo de caminho é propagado para todos os seus vizinhos. Este bit serve como sinal para a SC de que a IFP está propagando (ligado) ou parada (desligado) no momento.

O custo recebido pela IFP é processado pela sua unidade aritmética IDP e corresponde ao custo do caminho até a sua vizinha que o oferece. Então, primeiro, a IDP calcula o custo do caminho para si a partir do custo do caminho até a sua adjacente pela função de custo. Posteriormente, a IDP compara o seu custo atual com o custo processado. Caso o custo processado seja maior ou igual ao atual, nada é alterado em seus registradores. Caso contrário, seu custo e seu predecessor são atualizados e o seu bit do barramento *ready* é ligado. Além disso, o IM da IFP é avisado de que um novo custo precisa ser propagado para as IFP's adjacentes, iniciando a contagem de um ciclo de fases para a propagação.

A SC detecta o fim do processamento em uma fase em que todas as IFP's estão paradas, ou seja, com os bits do barramento *ready* desligados.

Durante o processo de propagação, o IM, indica qual o predecessor do IFT na fase atual. Caso o pixel seja conquistado seu predecessor é atualizado com este valor. A Tabela A.1 lista o predecessor na segunda e terceira colunas para cada fase da SIFT. A segunda coluna, é referenta ao uso de vizinhanças 4-conexa e a terceira coluna referente ao uso de vizinhança 8-conexa.

Como resultado de se propagar quaisquer custos de caminho assim que possível, tem-se caminhos não ótimos sendo propagados pelo grafo da imagem. Surgem então três questões relacionadas com este resultado:

- Vale a pena o processamento extra utilizado para propagar caminhos de custo não ótimo?

Fase	Predecessor 4-conexo	Predecessor 8-conexo
1	esquerda	esquerda
2	—	superior-esquerda
3	superior	superior
4	—	superior-direita
5	direita	direita
6	—	inferior-direita
7	inferior	inferior
8	—	inferior-esquerda

Tabela 5.1: Predecessores referentes à cada fase de processamento

O processamento extra utilizado para estes caminhos de custo não ótimo é sempre realizado por IFP's nas quais um caminho de custo ótimo ainda não chegou. Isto se dá porque, os pixels só são conquistados por caminhos de custo menores. Portanto, se algum caminho ótimo atingir um pixel este não pode ser conquistado novamente.

Estas IFP's que propagam caminhos de custo não ótimo ficariam ociosas, caso esperassem por um caminho de custo ótimo. Portanto, o processamento utilizado para caminhos não ótimos não interfere no processamento de caminhos ótimos.

- O algoritmo termina em algum instante?

  Esta questão pode ser respondida a base da função de custo de caminho utilizada pela SIFT. Como as funções utilizadas são monotonicamente incrementais, ($f_{\max}$ e f_{sum}), o custo do caminho só aumenta durante o seu percurso. As IFP's são conquistadas apenas por custos de caminhos menores. Portanto, um caminho que já passou por algum pixel não pode conquistá-lo novamente.

  Como as origens e pixels de percurso são finitos, os pixels não podem ser reconquistados indefinidamente por custos cada vez menores. Portanto, alguma configuração em algum momento é atingida de forma que todos os pixels já propagaram seu custo de caminho e não chega a eles um custo menor. Neste instante, o bit de estado de cada IFP' está desligado.

- O algoritmo alcançará uma solução ótima?

  Esta pergunta está relacionada com a corretude do algoritmo.

  Uma vez formado, o caminho ótimo não pode ser alterado, pois nenhum pixel que pertença a ele receberá um custo de caminho menor que o seu atual, sendo conquis-

tado. Logo, a única alternativa para não funcionar o algoritmo é, de alguma forma, o custo do caminho mínimo não chegar a algum pixel da imagem.

Pelo algoritmo da SIFT, porém, garante-se que uma IFP (pixel) sempre propaga seu custo de caminho a todas as suas adjacentes pelo seus mecanismos de ciclos de fases. Sendo assim, a partir de uma IFP (pixel), que contém um caminho mínimo, todas as IFP's adjacentes são atingidas.

A estrutura interna da SIFT garante também que toda IFP (pixel) atualize o caminho mínimo que recebe das suas adjacentes. Todo custo de caminho vindo de uma IFP adjacente é processado pela IDP. Caso o custo de caminho processado seja menor que o atual, então ele é atualizado nos seus registradores de custo de caminho e de predecessor. A propagação deste custo de caminho é iniciada logo a seguir, ligando-se o bit do barramento *ready* e com o IM contando as fases de propagação.

A garantia de que o processamento não pára enquanto algum pixel tiver que propagar seu custo pelo bit do barramento *ready* das IFP's (pixels), que estão propagando seu custo.

Logo, como os custos de caminho mínimo sempre são propagados e atualizados em todos os possíveis caminhos na imagem, sem serem reconquistados por outros custos de caminho e sem parar o processamento antes do tempo previsto, a floresta formada pela SIFT é de custos de caminhos mínimos.

Depois de processada a imagem, o mapa de custos de caminho da floresta gerada é transmitido para a interface externa. Depois, o mapa de predecessores contendo a floresta propriamente dita é transmitido e assim finaliza-se todo o processo.

Assim como na leitura de dados, a escrita é controlada pela SC, que possui contadores de linha e coluna dos pixels da imagem, para sair com os dados do endereço da IFP correta.

Durante o tempo de escrita, o barramento *run* permanece com valor '0', o *state*, o *ready*, o *we* e o *data_in* não são levados em conta, o *data_out* transmite os dados e o *re* assume o valor '01' quando um custo deve ser gravado e '10' quando o predecessor devem ser gravados. A Figura 5.10 ilustra a temporização das operações.

5.1.3 Implementação do Modelo Global

Validar e verificar a corretude de um modelo em hardware é uma tarefa bem complicada. Os recursos para depuração são escassos e limitados, além de demandar muito tempo, mesmo se tratando de uma FPGA. Por causa disso, foi adotada a estratégia de gerar a SIFT em 4 etapas baseado no Manual de Reuso [38].

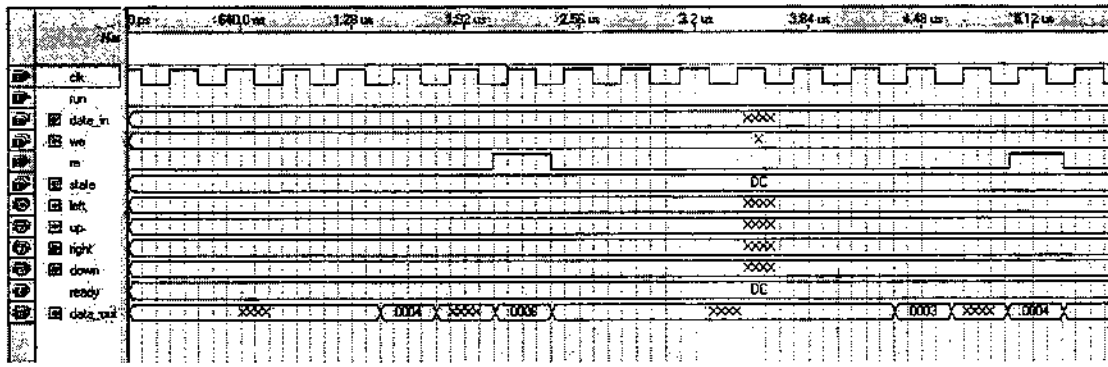


Figura 5.10: Comportamento dos sinais de entrada e saída de uma IFP durante a escrita de um valor de custo e um valor de predecessor, nesta ordem.

Na primeira etapa foi feita uma descrição bem detalhada da arquitetura, comportamento, temporização e sinais do modelo. Com esta descrição pôde-se verificar seu grau de dificuldade e estimar seu tempo de implementação. Ela serviu de guia durante todo o restante do projeto, principalmente para a segunda etapa.

Na segunda etapa, uma implementação em alto nível em linguagem C++ foi realizada para simulações comportamentais e funcionais do modelo proposto. A Figura 5.1 demonstra os resultados desta simulação, que foi bem sucedida.

Outra implementação foi realizada na terceira etapa, em linguagem SystemC. Esta linguagem é mais próxima da descrição de hardware e ajudou a refinar a temporização e os sinais, que o modelo em C++ não dá suporte. Apesar desta ajuda, a linguagem não suportou trabalhar com matrizes grandes de IFP's. Isto impediu a realização de testes com as imagens anteriores. No entanto, simulações com imagens pequenas (8x8 pixels) foram corretas.

Finalmente, durante a quarta etapa realizou-se a implementação em VHDL. Infelizmente esta implementação não foi viável devido ao tamanho da FPGA disponível para o projeto. Para uma imagem de 200x200 pixels a SIFT teria que possuir mais de 40.000 IFP's. Cada IFP ocupa em torno de 450 ALUT's (unidades geradoras de circuito na tecnologia Stratix-II). A FPGA Stratx-II EP2S60 possui 48.000 ALUT'S, dispondo de apenas duas para cada IFP. Por enquanto, a atual tecnologia das FPGA's não permite uma implementação deste tamanho.

Porém, as simulações em C++ apontam para ótimos resultados de tempo de processamento, caso a implementação em um hardware alternativo seja feita.

Foram contados o número de ciclos no processamento de algumas imagens. Pelo número de ciclos encontrou-se o número de fases que é multiplicado por 4 ou 8 dependendo da adjacência escolhida. Por fim, multiplicou-se o número de fases pelo período de clock, chegando-se ao tempo de processamento.

De acordo com as simulações, para processar uma imagem de 150x200 pixels, pela SIFT Global, são necessárias entre 250 e 800 fases. Se implementada para funcionar a uma freqüência de 50MHz, a operação seria realizada entre 5 e 16μs, enquanto que a mesma operação em um Athlon 64 de 3GHz com 512KBs de memória cache e 2GBs de memória RAM leva entre 8 e 13ms.

No entanto, como este modelo implementado em VHDL se mostrou demasiadamente grande em comparação com o tamanho das FPGA's disponíveis no mercado atualmente, pensou-se em um segundo modelo, descrito a seguir.

5.2 Modelo Setorial

O objetivo deste modelo em relação ao primeiro é de reduzir área sem perder muita performance. Logo, um modelo simples foi formulado, com base no primeiro.

A idéia é dividir a imagem em setores processados separadamente. Um setor é uma matriz retangular de pixels, que representa uma região da imagem. Todos os setores da imagem são do mesmo tamanho e possuem uma fileira de pixels comum a cada setor adjacente em sua vizinhança-4, conforme a Figura 5.11.

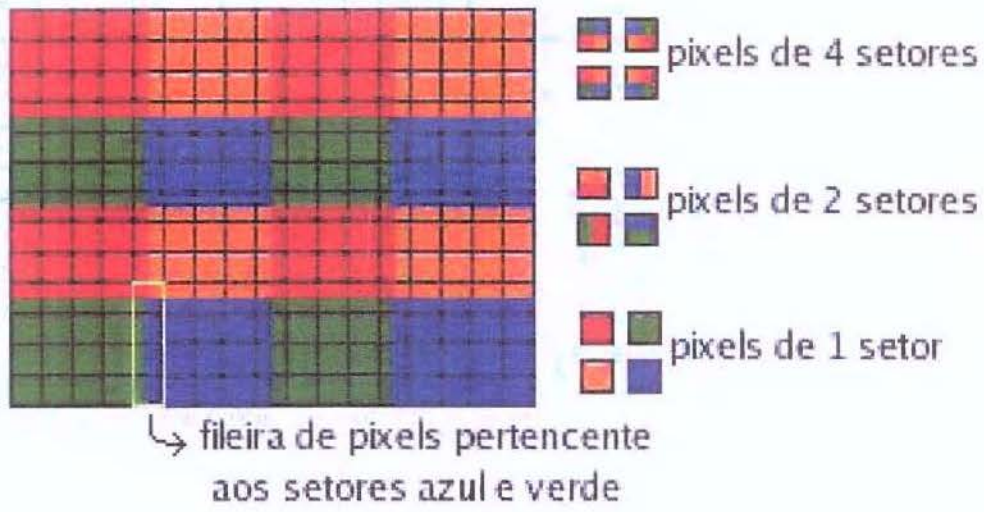


Figura 5.11: Imagem de 17x13 pixels dividida em setores de 5x4 pixels. Os setores são conjuntos de pixels adjacentes que contem uma mesma cor.

Com base no modelo global, existem algumas alterações na estrutura e funcionamento do modelo setorial conforme descrito a seguir.

5.2.1 Estrutura da SIFT no Modelo Setorial

A estrutura básica da SIFT deste modelo continua a mesma. Há uma matriz de IFP's e uma unidade de controle SC. Ambas, porém, tiveram algumas modificações.

O número de IFP's agora não é mais de uma por pixel da imagem, mas uma por pixel do setor da imagem. Os barramentos de ligação entre as IFP's são os mesmos acrescidos de um barramento que liga a IFP inicial com a final de cada fileira como na Figura 5.12.

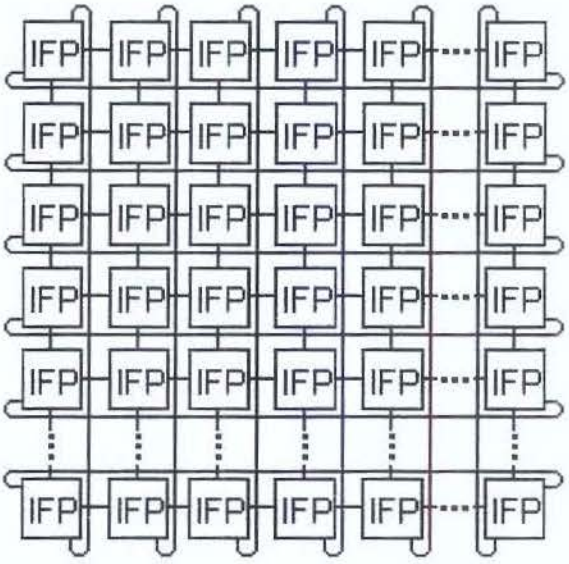


Figura 5.12: Matriz de IFP's da SIFT na solução setorizada. Além dos barramentos de ligação contidos na solução global, há também barramentos que ligam o primeiro ao último elemento de cada linha e coluna. O número de IFP's que a SIFT contém corresponde ao tamanho dos setores em que a imagem é dividida.

Decorrente do fato das IFP's cobrirem a área de um setor, a IR não possui mais apenas os dados de um pixel, mas de um pixel de cada setor cobrindo assim toda a imagem. Na Figura 5.13 ficam evidentes os pixels processados por cada IFP e percebe-se que um pixel é processado e tem seus dados armazenados dentro de apenas uma IFP.

Há também outra pequena mudança na IR. Ela possui neste modelo três bits de estado para cada pixel que representa ao invés de um.

Os barramentos de controle entre a SC e as IFP's também sofreram acréscimos. Além dos barramentos que já existiam, foram acrescentados os seguintes:

- uma entrada de 1 bit chamada *new_sector*. Por meio deste barramento a SC indica às IFP's que ouve alteração do setor processado. Há um barramento saíndo da SC para todas as IFP's.

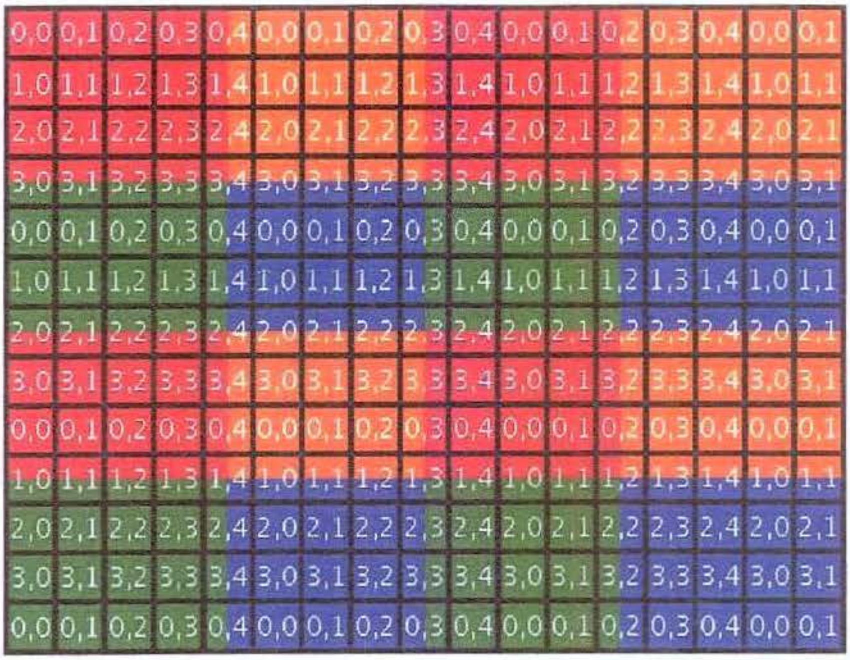


Figura 5.13: Imagem de 17x13 pixels dividida em setores de 5x4 pixels. O número em branco dentro de cada pixel indica a coordenada da IFP que contém os dados do mesmo.

- uma entrada de 3 bit chamada *addr*. Indica o endereço do pixel que a IFP deve processar. O endereço refere-se a um número de setor, portanto, a largura deste barramento é k, tal que 2^k representa o número de setores da imagem. Há um barramento *addr* da SC ligado a cada IFP, pois o endereço da memória IR acessada em um setor pode ser diferente para cada IFP.

- uma entrada de 4 bits chamada *lurd_neighbor_en*. Os bits deste barramento inibem o envio de dados do pixel do endereço dado por *addr* aos pixels vizinhos caso estejam desativados. O primeiro bit controla o fluxo da esquerda, o segundo o fluxo superior, o terceiro o fluxo da direita e o quarto o fluxo inferior. Um barramento por coluna de IFP's sai da SC referente aos vizinhos da esquerda e da direita e um barramento por linha de IFP's sai da SC referente aos vizinhos superior e inferior.

- uma entrada de 2 bits chamada *img_boarder*. Este barramento indica que o pixel do endereço dado por *addr* pertence a uma ou duas bordas da imagem. Um dos bits indica se o pixel pertence à borda superior ou inferior da imagem e o outro indica se o pixel pertence à borda da esquerda ou da direita da imagem, não sendo necessário distinguir em qual delas se encontra. Um barramento por coluna de IFP's sai da SC referente às bordas da esquerda e da direita e um barramento por linha de IFP's

sai da SC referente às bordas superior e inferior.

Os barramentos de dados permanecem inalterados. As Figuras 5.14 e 5.15 descrevem os novos barramentos utilizados.

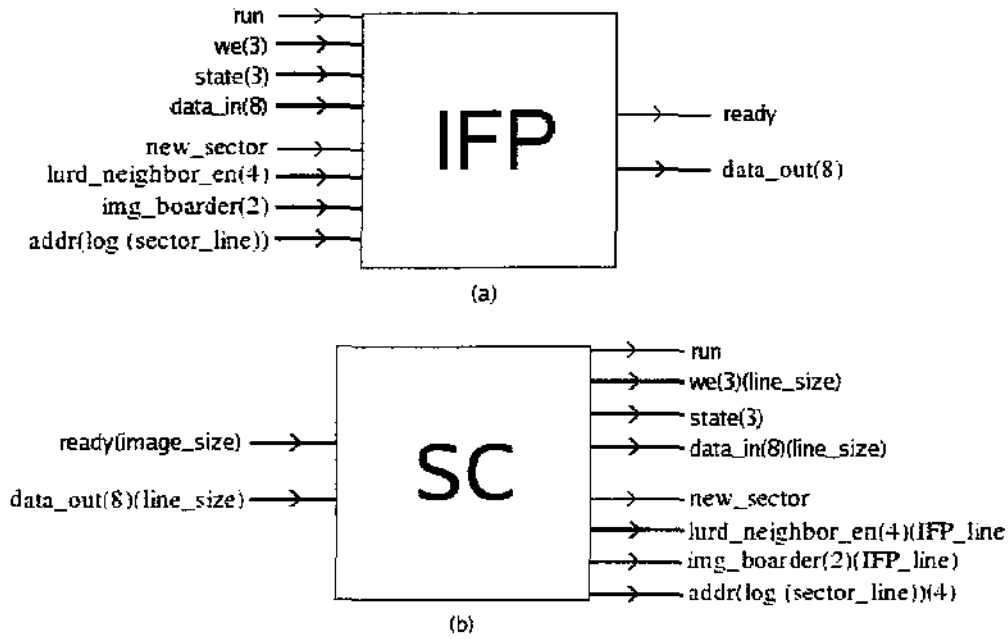


Figura 5.14: Roteamento de (a) uma IFP e (b) da SC com seus respectivos barramentos de entrada e saída. Os índices entre parênteses depois dos nomes dos barramentos indicam a largura dos mesmos, sendo de uma ou duas dimensões. Os barramentos representados por linhas finas são de apenas 1 bit.

A SC têm os mesmos controles do modelo global, além de controlar também o setor a ser processado pelas IFP's. A condição de parada sofreu uma leve mudança.

As razões das mudanças estruturais são descritas a seguir, no funcionamento do modelo setorial.

5.2.2 Funcionamento da SIFT no Modelo Setorial

A entrada e a saída de dados na SIFT Setorial são realizadas da mesma maneira que na SIFT Global. Dentre os novos barramentos da SIFT Setorial, o único utilizado nesta etapa é o *addr*, que indica em que endereço de memória a IFP deve gravar e ler os dados. Veja as formas de onda de transmissão de dados, na SIFT Setorial, nas Figuras 5.16 e 5.17.

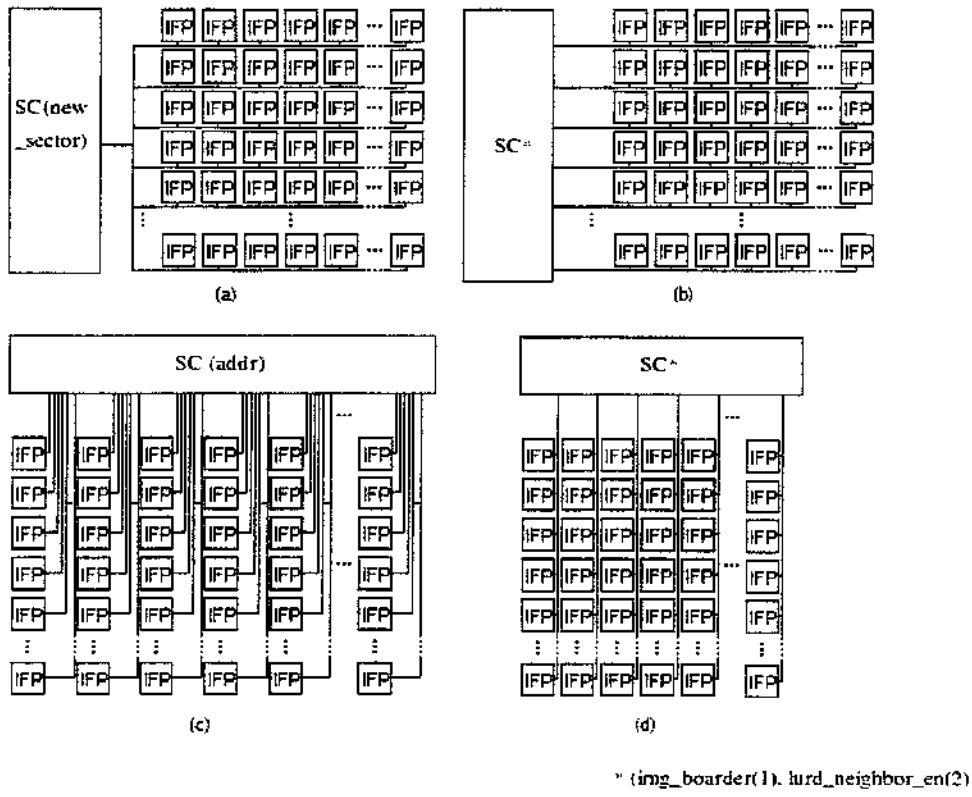


Figura 5.15: Barramentos entre a SC e a matriz de IFP's. Configuração dos barramentos (a) *new_sector* (b) 1 bit do *img_boarder* e 2 bits do *lurd_neighbor_en* (c) *addr* e (d) bits restantes do *img_boarder* do *lurd_neighbor_en*.

O processamento, por sua vez, teve alterações em seu funcionamento básico. Além de fases e ciclos, na SIFT Setorial há também varreduras.

Os conceitos de fase e ciclo continuam os mesmos. Em cada fase uma IFP pode enviar seu custo de caminho para uma adjacente e receber de outra, sendo ou não conquistada. Um ciclo é uma seqüência de fases nas quais uma IFP pode passar seu custo de caminho para cada IFP adjacente e também receber dados de cada uma delas, sendo ou não conquistada.

A seqüência com que os setores são escolhidos é em linhas da esquerda para a direita, a partir da linha superior até a inferior, recomeçando quando a última linha de setores é percorrida. O percurso de todos os setores da imagem é chamado de varredura.

No entanto, o número de IFP's é menor. Com isso, um setor da imagem de cada vez é escolhido para ser processado pelo mesmo método da SIFT Global, ou seja, por fases e ciclos.

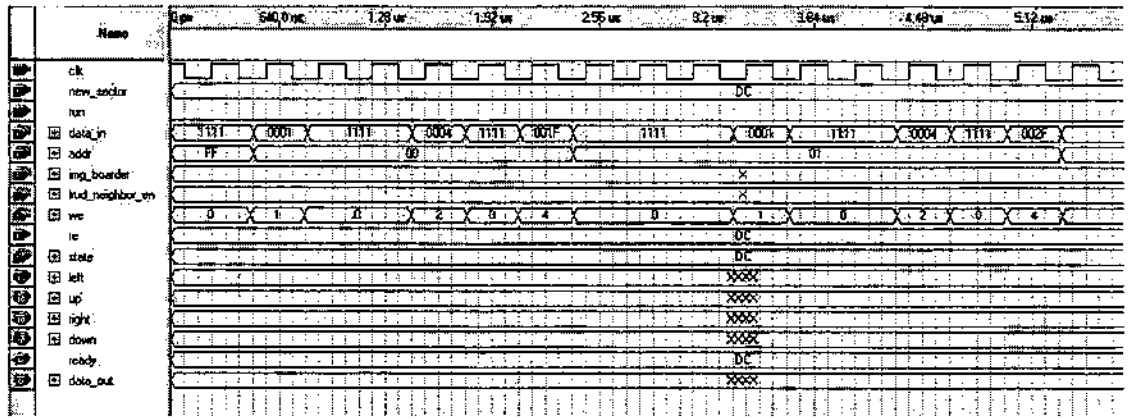


Figura 5.16: Comportamento dos sinais de entrada e saída de uma IFP durante a leitura de um valor de brilho, um valor de custo e um valor de predecessor, vizinhança e bits semente, nesta ordem.

Há mudanças quanto ao início da propagação de custos de caminho. Os pixels começam a propagação não mais na subida do bit do barramento *run*, mas na subida do bit do barramento *new_sector* e o fazem caso os bits sementes possuem valor maior que 0.

Durante a entrada de dados a responsabilidade é do usuário de entrar com o valor correto dos bits sementes para cada pixel que é semente, dependendo de a quantos setores cada pixel pertence.

Por exemplo, um pixel semente que pertence a quatro setores tem o valor quatro em seus bits sementes antes do início do processamento. Depois de ter transmitido o seu custo em um dos setores, este pixel semente terá o valor três nos bits de estado, pois faltam três outros setores para ele propagar seu custo.

Como visto anteriormente, as ligações entre as IFP's estão agora acrescidas de barramentos entre as IFP's do início e fim de cada fileira como na Figura 5.12.

Estes barramentos são necessários na SIFT Setorial, porque as IFP's do início e do fim das fileiras podem processar pixels que são adjacentes, tendo que transmitir custos de caminho entre si.

Escolhendo-se o setor vermelho da primeira coluna e da primeira linha da imagem na Figura 5.13, a primeira coluna de pixels deste setor é processada pela primeira coluna de IFP's. A última coluna de pixels é processada pela última coluna de IFP's

Escolhendo-se o setor laranja da segunda coluna e primeira linha da imagem na Figura 5.13, percebe-se que a primeira coluna de pixels deste setor não está sendo processada pela primeira coluna de IFP's, mas pela última. A segunda coluna de pixels é processada pela primeira coluna de IFP's e a última coluna de pixels é processada pela penúltima coluna de IFP's.

Neste caso, deve existir uma ligação entre a primeira e a última coluna de IFP's e esta

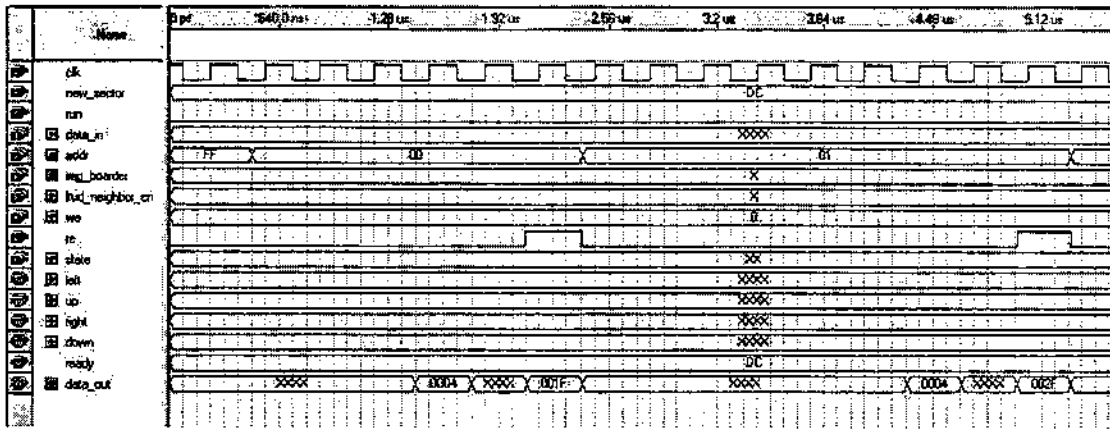


Figura 5.17: Comportamento dos sinais de entrada e saída de uma IFP durante a escrita de um valor de custo e um valor de predecessor, nesta ordem.

ligação não deve existir entre a primeira e a segunda coluna. Portanto, dependendo do setor escolhido, uma das linhas e uma das colunas de barramentos é aberta pela SC, não permitindo o tráfego de custos de caminho. A inibição ou ativação destes barramentos entre as IFP's propriamente ditas é realizada pela IDM com base no controle proveniente dos barramentos *lurd_neighbor_en* e *img_boarder*.

Como dito na seção anterior, estes barramentos estão divididos entre as linhas e colunas das IFP's. No máximo, um pixel fará parte de duas bordas da imagem e no mínimo terá dois vizinhos ativos em cada setor a que pertence.

Os pixels que pertencem a quatro setores, por exemplo, como mostra a Figura 5.11, sempre terão dois dos qutro bits do barramento *lurd_neighbor_en* ligados e dois desligados, além disso sempre possui os bits do barramento *img_boarder* desativados.

Já os pixels que pertencem a dois setores podem ou não fazer parte de um borda da imagem, tendo sempre um bit do barramento *img_boarder* desativado. O outro bit estará ativado caso ele pertença à borda da imagem.

Finalmente, se um pixel pertence a apenas um setor e não faz parte da borda da imagem, então seu barramento *img_boarder* está desativado e seu barramento *lurd_neighbor_en*, está ativado. Se pertencer à uma borda um bit de *img_boarder* é ativado e um de *lurd_neighbor_en* é desativado. Caso pertença a um canto da imagem, seu barramento *img_boarder* está desativado e seu barramento *lurd_neighbor_en*, está com dois bit ativado.

Além de controlar o fluxo de dados entre IFP's, o barramento *lurd_neighbor_en* ajuda também no controle dos bits sementes. Se um pixel t pertence aos setores w_1 e w_2 e é conquistado enquanto o setor w_1 é processado, então ele precisa propagar o novo custo de caminho dentro do próprio setor, e também para seus vizinhos no setor w_2. A imagem é feita, a SC conta o número de setores consecutivos quessim que a conquista de t é

realizada, a IM da IFP grava em seus bits sementes o valor 1, indicando que ele precisa propagar seu custo em mais um setor.

A Figura 5.18 ilustra o comportamento da IFP na SIFT setorial, durante o processamento de dados.

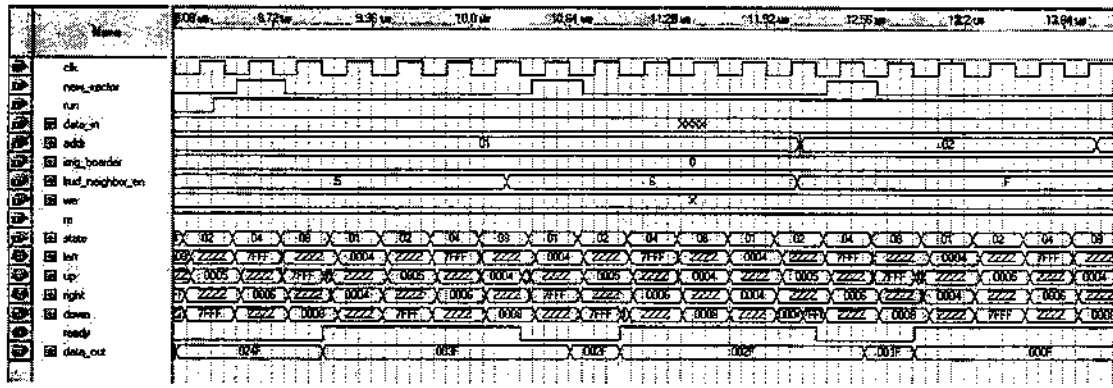


Figura 5.18: Comportamento dos sinais de entrada e saída de uma IFP durante o processamento da SIFT setorial. As formas de onda compreendem um trecho de simulação que iniciam-se com um ciclo em que os dados de um pixel do canto superior direito são propagados aos vizinhos. Depois disso, a SC muda para o setor seguinte e a IFP passa a escolher o mesmo pixel, porém agora sendo do canto superior esquerdo. Ele transmite seus dados aos vizinhos e a SC mais uma vez muda de setor. Neste instante porém, escolhece um pixel que pertence ao centro do setor. Este pixel propaga seu custo e termina o processamento.

Foi visto também, na Seção 5.1.2, que em algum instante o processamento da imagem na SIFT Global pára. Isto ocorre quando todos os pixels da imagem não precisam mais propagar seu custo. Se encararmos a imagem como um setor grande, o mesmo princípio se aplica para um setor pequeno. No instante em que o processamento de um setor pára, outro setor é escolhido para ser processado.

Os bits sementes são lidos pela SC através do barramento *data_out*, durante o processamento. Enquanto a varredura da imagem é feita, a SC conta o número de setores consecutivos que possuem todos os bits semente de todos os seus pixels com valor 0. Assim, o processamento da imagem termina quando a SC checa, durante uma varredura completa, que todos os setores contém todos os bits sementes com valor 0.

Surgem novamente perguntas para a SIFT Setorial:

- O algoritmo termina em algum instante?
- O algoritmo alcançará uma solução ótima?

A resposta a estas duas perguntas é a mesma dada para a SIFT Global. O que difere um modelo do outro é apenas a ordem de propagação.

O sistema de setores continua não permitindo que custo de caminhos maiores conquistem algum pixel, pois os cálculos utilizados através das funções de custo de caminho das IDP's são exatamente os mesmos. Também as possíveis sementes e caminhos continuam sendo finitos. Portanto o processamento pára em algum instante.

O caminho de custo mínimo também é sempre propagado. Os pixels do meio de setores propagam os custos da mesma forma que no modelo global. Os pixels da borda dos setores garantem a propagação dos seus custos para todos os adjacentes pelos bits de estado extras e pelo sistema de varredura utilizados neste modelo.

Além disso, o estado interno dos pixels do setor processado e os bits de sementes impedem que o processamento pare antes da propagação de algum custo de caminho.

Assim, a floresta encontrada neste modelo também é ótima.

5.2.3 Implementação da SIFT no Modelo Setorial

A implementação da SIFT Setorial foi baseada diretamente na implementação da SIFT Global em VHDL, pois a maioria dos controles e dados são os mesmos salvo à divisão da imagem em setores.

A área utilizada pela SIFT Setorial na FPGA é diretamente proporcional ao tamanho do setor, pois o tamanho do setor é exatamente o número de IFP's. Sendo assim, a SIFT Setorial foi projetada para que o tamanho do setor seja facilmente configurável, de modo a poder reutilizar a arquitetura ao passo que a tecnologia das FPGA's progride.

Como visto na Seção 2.4, encontrar uma interface que atenda aos requisitos do sistema é um grande desafio, principalmente quando uma grande quantidade de dados é requerida em pouco tempo. Sua escolha depende também da arquitetura disponível para a implementação.

O Laboratório de Sistemas de Computação do Instituto de Computação da Unicamp disponibilizou uma placa Stratix-II modelo EP2S60F672E5 para o projeto. Esta placa tem como alternativas de Entrada e Saída 2 portas seriais, um conector de rede ethernet RJ45 com chip que suporta fluxos 10/100mbps, um conector de cartão de memória flash (CFC) e pinos de entrada e saída.

Como visto também na Seção 2.4, uma interface específica para uma destas alternativas seria mais rápida e menor que utilizar um processador Nios-II. O professor Elmar Melcher da Universidade Federal da Paraíba, fez uma grande contribuição neste sentido, cedendo um código base para a implementação de leitura e escrita no cartão de memória flash, conforme a especificação [39].

Este cartão permite leitura e escrita de dados em barramentos 8/16 bits. Portanto, a

SIFT foi planejada para operar o fluxo de dados por barramentos desta largura. O cartão de memória flash (CF Card) utilizado possui 15.1MB.

Para maior simplicidade a SIFT foi implementada tendo o mesmo número de linhas e colunas. O número máximo de IFP's, em termos de elementos lógicos, que a placa Stratix-II EP2S60F672C5 suporta é de 12x12, com o código compilado no programa Quartus-II Versão 5.1. No entanto, com uma matriz de 12x12 IFP's, a FPGA não suportou 144 memórias implementadas entre os blocos M512RAM, M4KRAM e M-RAM disponíveis da placa. Se utilizados elementos lógicos para as memórias o espaço não é suficiente.

Portanto, uma matriz de 11x11 IFP's atendeu a ambos os requisitos de elementos lógicos e blocos de memória na placa. O número de setores que a imagem contém também foi escolhido de acordo com a placa. A Stratix-II suportou uma matriz de 20x20 setores. Portanto, o número de pixels da maior imagem possível é de 220x220 pixels.

O tempo gasto com a entrada e saída dos dados, para uma imagem de 220x220 pixels, na atual interface, é da ordem de 120ms. O tempo de transmissão é equivalente ao número de leituras realizadas multiplicado pelo tempo de leitura somado ao número de escritas realizadas multiplicado pelo tempo de escrita da memória flash. O tempo de uma leitura é de 14 ciclos de clock e o de uma escrita é de 24 ciclos de clock numa de freqüência de 50MHz. O número de transferências é de $2 \times 211 \times 211$ para o custo e para o brilho e 211×211 para o predecessor, o que dá um total de $5 \times 211 \times 211$ leituras e $3 \times 211 \times 211$ escritas.

Esta lentidão se deu porque a memória flash tem um atraso muito grande para gravar dados, além de possuir um barramento estreito de no máximo de 16 bits. No entanto há boas alternativas para reduzir este tempo, que serão discutidas no próximo capítulo.

As Figuras 5.7, 5.8, 5.10, 5.16, 5.18 e 5.17 apresentadas neste capítulo foram todas adquiridas através de simulações do código VHDL implementado da SIFT Setorial. Por meio delas, a corretude do algoritmo pode ser observada. Os sinais de entrada e saída também estão muito bem comportados, como o esperado.

O tempo de execução da SIFT Setorial é calculado pelo número de ciclos de processamento necessários. Os ciclos na vizinhança 8-conexa duram 8 fases, ou 8 ciclos de clock, e na vizinhança 4-conexa duram 4 fases, ou 4 ciclos de clock como visto na Seção 5.1.2 e freqüência máxima de operação do circuito gerado pela ferramenta Quartus-II da Altera é de 7,60MHz. Assim, o circuito final se mostrou satisfatório tanto no tamanho máximo da imagem processada, quanto na sua velocidade.

Capítulo 6

Conclusões

A SIFT, que é uma implementação paralela em hardware do algoritmo da IFT, se mostrou uma ótima ferramenta no processamento de imagens. Ela foi desenvolvida em duas implementações: a SIFT Global, na qual cada pixel é processado por uma unidade chamada IFP; e a SIFT Setorial, na qual a imagem é dividida em setores com processadores para cada pixel contido neles.

O modelo da SIFT Global se mostrou ideal para o processamento de imagens. Em comparação com a IFT em software, a SIFT Global opera cerca de 1000 vezes mais rápido.

A SIFT Setorial, por sua vez, mostrou-se uma implementação adequada para a FPGA disponível para o projeto, apesar do grande atraso na transferência de dados. Este problema está muito mais relacionado com a plataforma de desenvolvimento do que com o algoritmo da SIFT em sí, pois, a SIFT suporta a entrada/saída simultânea dos dados de uma linha inteira de IFP's.

Ela pode ser considerada um modelo para o processamento de imagens em hardware devido a sua estruturação, ao utilizar o paralelismo de imagem, tendo a perda de apenas um ciclo de clock na mudança do setor utilizado pelos processadores. Dessa forma, cada IFP ficou responsável pelo processamento de um conjunto fixo e distinto de pixels. Elas não necessitam de grandes trocas de contexto, já que os dados necessários para cada IFP estão em suas memórias internas e não são utilizados por outras.

Para o processamento de imagens, a SIFT se mostrou muito útil em diversos dos operadores da IFT original. As principais aplicações dela são de segmentação de objetos por percurso de bordas e crescimento de regiões e cálculo de distâncias geodésicas utilizando distâncias de Chanfer.

Durante o trabalho, ficou mais claro também o conceito do mapa de predecessores inicial para a IFT, não sendo necessário um mapa trivial para se atingir um resultado correto. Isto fica evidente pela própria estrutura da SIFT que não utiliza dados de predecessores como meio de gerar a floresta.

Evidenciou-se também, que a reconquista de pixels, como é realizada na DIFT [3] e na SIFT, pode gerar resultados corretos, desde que as devidas precauções sejam tomadas, como a ordenação da fila de prioridade e escolha de sementes, no primeiro caso, e critérios de propagação e parada bem definidos, no segundo caso.

6.1 Trabalhos Futuros

A SIFT Global ou a Setorial podem ser implementadas em outras plataformas de hardware para diminuir principalmente o tempo de entrada e saída dos dados e aumentar o tamanho do setor e da imagem processada em ambos os modelos.

Uma das alternativas é de utilizar uma FPGA com acesso por porta paralela, ou por barramento PCI, o que ampliaria a largura da banda de transferência. A FPGA também pode ser de um modelo mais recente, que opere a freqüências maiores e que disponha de mais elementos lógicos e memória.

Uma outra alternativa é implementar um dos modelos em um ASIC a partir do modelo gerado para a FPGA, que pode operar a freqüências bem maiores, como 400MHz com uma tecnologia $1.8\mu m$ [40] e barramentos largos para entrada e saída de dados.

Uma outra proposta que pode ser feita dentro da plataforma atual é de mudar a interface para que uma mesma imagem possa ser processada diversas vezes sem ter que entrar e sair com todos os dados toda vez que um resultado é obtido. Pode-se implementar uma interface que permita alterar apenas alguns valores de custo, predecessor, ou sementes, entre processamentos, semelhante ao realizado na DIFT [3]. Um comando de reset ou set, também poderia ser implementado para colocar um determinado valor inicial de custo e predecessores em toda a imagem, quando estes valores ocorrem na grande maioria dos pixels.

A varedura da imagem também poderia ser melhor explorada. Alternando a direção da mudança de setores, a propagação dos custos seria facilitada em todas as direções.

Pode-se, também, depois de integrar a plataforma da SIFT a um computador, criar uma interface de software para dar comandos diretos à SIFT e obter os resultados de operadores através de um programa. Isto poderia transformar a SIFT em uma plataforma educacional, ou parte de um sistema maior.

Por exemplo, pode-se criar um pipeline para detecção automática de objetos, utilizando aplicações exaustivas da SIFT, para encontrar a localização mais provável do objeto na imagem. A maioria dos métodos de detecção atuais requer algum tipo de interação com o usuário [3, 13, 41–43] e a utilização da SIFT seria uma revolução neste sentido.

Apêndice A

Tabela de Símbolos

Símbolo	Significado
$\mathcal{I}$	Uma imagem.
$\mathbf{I} = (\mathcal{I}, I)$	Imagem representada como grafo, onde $\mathcal{I}$ representa os pixels como vértices e I representa os brilhos dos vértices.
s, t	Pixels ou vértices de um grafo representando uma imagem.
(s, t)	Aresta do vertice s para o vertice t em um grafo representando uma imagem.
$\mathcal{A}$	Relação de adjacência utilizada para a representação de uma imagem como grafo.
$d(s, t)$	Distância entre os pixels s e t dentro de uma imagem.
π, τ	Caminhos em um grafo que representa uma imagem.
$\pi \cdot \tau$	Concatenação dos caminhos π e τ com o vértice de ligação entre eles em comum.
$f(\pi)$	Função de custo do caminho π.
$\mathcal{V}$	Conjunto universo de custos de caminhos.
$w(s, t)$	Peso da aresta (s, t) dado por uma função de custo f.
$h(t)$	Custo inicial de caminho trivial $\langle t \rangle$
$I(t)$	Brilho do pixel t na imagem $\mathcal{I}$
$P(t)$	Predecessor do pixel t na imagem $\mathcal{I}$
$R(t)$	Raiz do pixel t na imagem $\mathcal{I}$
$\mathcal{F}, \mathcal{Q}$	Conjuntos de vértices(pixels).
$\mathcal{S}$	Conjunto de sementes utilizados em uma IFT.
$\mathcal{Q}$	Fila de prioridade Circular da IFT.

Tabela A.1: Símbolos definidos nos capítulos 3 e 4

Referências Bibliográficas

[1] A. X. Falcão, J. Stolfi, and R. A. Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(1):19–29, 2004.

[2] R.A. Lotufo and A.X. Falcão. The ordered queue and the optimality of the watershed approaches. In *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, pages 341–350. Kluwer, Jun 2000.

[3] A.X. Falcão and F.P.G. Bergo. Interactive volume segmentation with differential image foresting transforms. *IEEE Trans. on Medical Imaging*, 23(9):1100–1108, 2004.

[4] A.X. Falcão, F.G. Bergo, and P.V. Miranda. Image segmentation by tree pruning. *In Proceedings of the XVII Brazilian Symposium on Computer Graphics and Image Processing, Curitiba, PR*, pages 65–71, Oct 2004.

[5] A.X. Falcão, P.A.V. Miranda, and F.P.G. Bergo. Automatic object detection by tree pruning. Technical Report IC-05-19, Institute of Computing, State University of Campinas, 2005.

[6] P.A.V. Miranda, R.S. Torres, and A.X. Falcão. TSD: A shape descriptor based on a distribution of tensor scale local orientation. In *XVIII Brazillian Symposium on Computer Graphics and Image Processing*, pages 139–146, Natal, RN, Oct 2005. IEEE Press.

[7] A.X. Falcão, B.S. da Cunha, and R.A. Lotufo. Design of connected operators using the image foresting transform. *In Proc. of SPIE on Medical Imaging*, 4322:468–479, Feb 2001.

[8] A.X. Falcão and B. S. da Cunha. Multiscale shape representation by the image foresting transform. In *Proceedings of SPIE on Medical Imaging*, volume 4322, pages 1091–1100, San Diego, CA, Feb 2001.

[9] A.X. Falcão, L.F. Costa, and B.S. da Cunha. Multiscale skeletons by image foresting transform and its applications to neuromorphometry. *Pattern Recognition, ISSN:0031-3203, Elsevier Press*, 35(7):1569–1580, 2002.

[10] R.S. Torres, A.X. Falcão, and L.F. Costa. Shape description by image foresting transform. In *14th International Conference on Digital Signal Processing*, Santorini, Greece, Jul 2002.

[11] R. A. Lotufo, A. X. Falcão, and F. Zampirolli. IFT-Watershed from gray-scale marker. In *Proc. of XV Brazilian Symp. on Computer Graphics and Image Process ing*, pages 146–152. IEEE, Oct 2002.

[12] R.S. Torres, A.X. Falcão, and L.F. Costa. A graph-based approach for multiscale shape analysis. *Pattern Recognition*, 37(6):1163–1174, 2004.

[13] A.X. Falcão, J.K. Udupa, S. Samarasekera, S. Sharma, B.E. Hirsch, and R.A. Lotufo. User-steered image segmentation paradigms: Live-wire and live-lane. *Graphical Models and Image Processing*, 60(4):233–260, Jul 1998.

[14] A.X. Falcão, J.K. Udupa, and F.K. Miyazawa. An ultra-fast user-steered image segmentation paradigm: Live-wire-on-the-fly. *IEEE Transactions on Medical Imaging*, 19(1):55–62, Jan 2000.

[15] A.X. Falcão and J.K. Udupa. A 3D generalization of user-steered live wire segmentation. *Medical Imaging Analysis*, 4(4):389–402, Dec 2000.

[16] David Ross, Oliver Vellacott, and Martin Turner. An fpga-based hardware accelerator for image processing. In *Selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs*, pages 299–306, Oxford, UK, UK, 1994. Abingdon EE&CS Books.

[17] Christof Koch Ffrank Perez. Toward color image segmentation in analog vlsi: Algorithm and hardware. *International Journal of Computer Vision(Historical Archive)*, 12(1):17–42, 1994.

[18] Moncef Gabbouj Alina N. Moga. Parallel image component labeling with watershed transformation. *IEEE Transactions on Pattern Analisys and Machine Inteligence*, 19(5):441–450, May 1997.

[19] Santanu Dutta da Philips Semiconductors. Arquitecture and implementation of multi-processor soc for advanced set-top box and digital tv systems. In *16th Symposium on Integrated Circuits and Systems Design*, pages 145–146, São Paulo, Brazil, Aug 2003.

[20] IEEE. (1076-2002) ieee standard vhdl language reference manual. available at http://standards.ieee.org/reading/ieee/std/dasc/1076-2002.pdf.

[21] IEEE. (1364-2001) ieee standard description language based on the *verilog*^{TM} hardware description language. available at http://standards.ieee.org/reading/ieee/std/dasc/1364-2001.pdf.

[22] F. Meyer. Topographic distance and watershed lines. *Signal Processing*, 38:113–125, 1994.

[23] H. Oswald S. Wegner, T. Harms and E. Fleck. The watershed transformation on graphs for the segmentation of ct images. *13th International Conference on Pattern Recognition (ICPR'96)*, 3:498, 1996.

[24] P. T. Jackway. Gradient watersheds in morphological scale-space. *IEEE Transactions on Image Processing*, 5(6):913–921, 1996.

[25] A. Wright and S. Acton. Watershed pyramids for edge detection. In *IEEE International Conference on Image Processing*, pages II:578–xx, 1997.

[26] R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(4):532–550, 1987.

[27] Noel A C Cressie Jean Paul Serra. *Image analysis and mathematical morphology.* Academic Press, London ; New York, 1982.

[28] A. Bernasconi, S.B. Antel, D.L. Collins, N. Bernasconi, A. Olivier, F. Dubeau, G.B. Pike, F. Andermann, and D.L. Arnold. Texture analysis and morphological processing of magnetic ressonance imaging to assist detection of focal cortical dysplasia in extra-temporal partial epilepsy. *Ann. Neurol.*, 49:770–775, 2001.

[29] P. E. Danielson S. Levialdi. Computer architectures for pictorial information systens. In *IEEE Computer Magazine*, pages 53–67, 1981.

[30] George S. Almasi and Allan Gottlieb. *Highly parallel computing (2nd ed.).* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[31] W. Stallings. *Arquitetura e organizacao de computadores: projeto para o desempenho.* Prentice Hall, São Paulo, SP, Brazil, 2002.

[32] Altera Corporation. *Nios Development Board Reference Manual, Stratix II Edition*, 2005. `http://www.altera.com/literature/manual/mnl_nios2_board_stratixII_2s60.p`

[33] U. Montanari. On the optimal detection of curves in noisy pictures. *ACM Press*, 14:335–345, May 1971.

[34] A. Martelli. Edge detection using heuristic search methods. *Computer Graphcs and Image Processing*, 1:169–182, 1972.

[35] A. Martelli. An application of heuristic search methods to edge and countor detection. *Com. ACM*, 19(2):73–83, May 1976.

[36] E.N. Mortensen and W.A. Barrett. Interactive segmentation with intelligent scissors. *Graphical Models and Image Processing*, 60:349–384, 1998.

[37] B.J.H. Verwer, P.W. Verbeek, and S.T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4):425–429, 1989.

[38] Michael Keating and Pierre Bricaud. *Reuse Metodology Manual.* Kluwer Academic Publishers, Boston/ Dordrecht/ London, 1999.

[39] CompactFlash Association. *CF+ and CompactFlash Specification Revision 3.0*, 2004. `http://lad.dsc.ufcg.edu.br/fenix/PDFs/cfspc3_0.pdf`.

[40] LSI-Logic. *G12 ASIC cell-based product*, 1999. `http://www.lsilogic.com/products/asic_technology/`.

[41] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 1987.

[42] L.D. Cohen. On active contour models and ballons. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 53(2):211–218, 1991.

[43] Marie-Pierre Jolly Yuri Y. Boykov. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Intl. Conf. on Computer Vision (ICCV)*, volume 1, pages 105–112, 2001.